Personal Computer

PC-8801 NEC mkII

データファイル入門[1]

ナツメ社編集部＝編著

ナツメ社



Personal Computer
PC-8801mkII
NEC
データファイル入門[1]
ナツメ社編集部＝編著
ナツメ社

[本書を読まれる方へ]

⦿パソコンをビジネスその他、
実務面で役立てようと思うと、
どうしてもディスク操作のマスターを迫られます。
しかしディスクでのデータの扱いは、
ベーシックのときと考え方を多少異にするため、
すぐにはなじめない一面があります。
⦿これが、ディスクはむずかしいといわれる
ゆえんだと思いますが、この理由のために、
途中で挫折してしまうか、
あるいはせっかくディスクの備えがありながら、
これをプログラムのセーブとロードのためだけに
使用することにとどまっている人が多いようです。
⦿要するに、多くのディスクが
ユーザーの手にわたっていながら、
その本来の使命が十分果たされていないようです。
⦿そこで、ベーシックとディスクベーシックとの
連絡をつけ、みなさんができるだけ抵抗なく
入っていけるようにと考え、発行されたのが本書です。
⦿ディスクは、あなたの努力次第で
ずいぶん実務面で活用できます。
⦿せっかくのディスク
なんとか活かしてみませんか?

cover design――nobuo nakagaki+yoshifumi hayase

Personal Computer

PC-8801

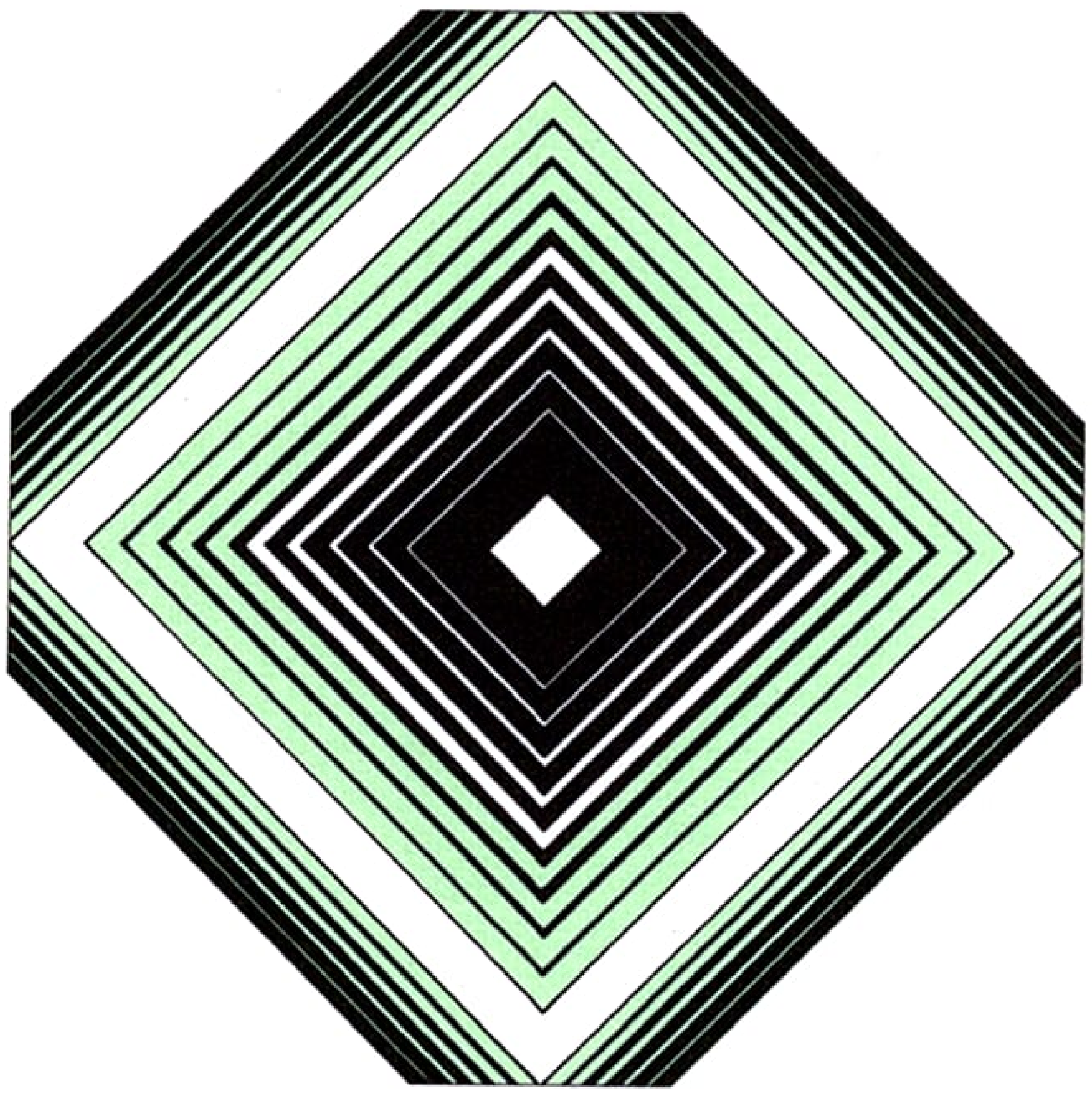
NEC

mkⅡ

データファイル入門[1]

ナツメ社編集部＝編著

ナツメ社



まえがき

この本は書名にもあるとおり、ＮＥＣのパソコン

ＰＣ—8801mk II

に関するディスク操作を解説したものです。

ビジネスなど実務面でたくさんのデータを扱おうと思うと、どうしてもディスクにたよらざるを得ません。

そこで、いままでベーシックで組んでいたプログラムをディスクバージョンに変更しなければならないわけですが、ディスクでのデータの扱いは考え方を多少異にするため、すぐになじめない一面があります。そして、この理由のために、途中で挫折してしまうか、あるいはせっかくディスクを持ちながら、これをプログラムのセーブとロードのためだけに使用することにとどまっている人が多いようです。

そこで編集部では、ベーシックとディスクベーシックとの連絡をつけ、みなさんができるだけ抵抗なく入っていけるようにと考え、本書を発行しました。

ディスクは、あなたの努力次第でずいぶん実務面で活用できます。せっかくディスクがあるのですから、これを活かしてみませんか？実行画面を多くかかげましたので大変わかりやすくなっています。

なおＰＣ—8801mk IIには、次の３つのタイプがあります。

❶ モデル10（ディスク装置なし）
❷ モデル20（１ドライブのディスク装置内蔵）
❸ モデル30（２ドライブのディスク装置内蔵）

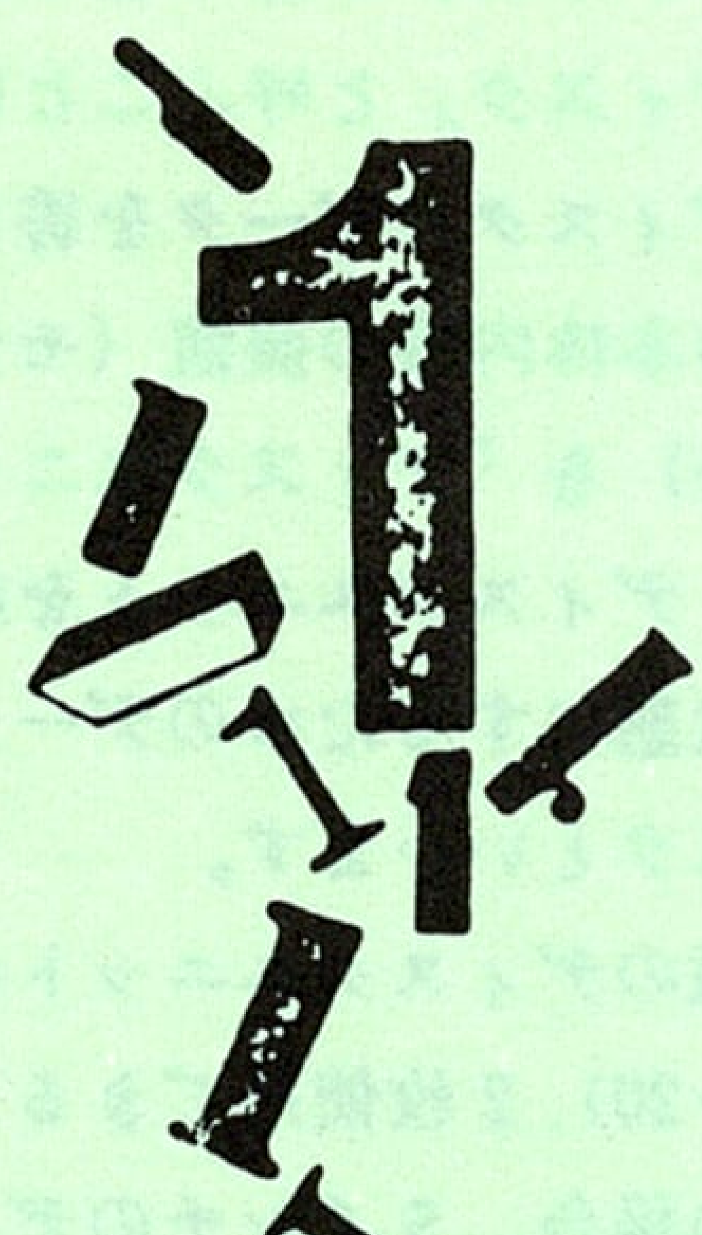
本書は、このうち❸のタイプを使って解説を進めています。❶の機種をお持ちの方は、ディスクユニットを購入し外部装置として取り付けてください。また、モデル20をご使用の方は、ドライブ２を使えませんので、本文中のファイルディスクリプタのドライブ番号指定を取り除いて読んでいただくことになります。

また、ディスク立ち上げ時の初期画面は、機種によって多少異なりますが、本書で扱うプログラムはどの機種でも正常に動作しますので変更は不要です。本書では、ベーシックの基本的なコマンド等には説明を加えておりませんので、プログラミングの入門編を終えてない方は、『ＰＣ—8801mk II　プログラミング入門』（ナツメ社）を読んでから、本書にお進みください。

編集部

もくじ

第1章
ディスクの基本操作



1・1　操作する前に

この章では、ディスクの基本的な操作方法について学びますが、本題に入る前に、いくつか注意を述べておきます。

フロッピィディスク

まず、ディスクのことを正式にはフロッピィディスクといい、記憶容量の違いにより、5インチミニフロッピィディスク、8インチ標準フロッピィディスクがありますが、**この本では**「ディスク」または「フロッピィディスク」と呼ぶことにします。

ディスクユニット

また、ディスクにデータを書き込んだり、あるいはそれを読んだりするための本体内蔵の装置（モデル20、30の場合）、外部装置（モデル10の場合）を「ディスクユニット」と呼ぶことにします。

システムディスク

そして、ディスクユニットを起動し、ディスクへのオペレーションができる状態にするためのデータが書き込まれているディスクをシステムディスクといいます。

本体内蔵のディスクユニットの場合、ディスクを1枚使用できるもの（モデル20)、2枚使用できるもの（モデル30）の2種類が、また、モデル10の場合、5インチのディスクを使うか8インチのディスクを使うかによってディスクユニットも分かれますが、実質的な操作方法には違いがありません。

本書では、モデル30を使用して説明を進めていますので、ディスクのコピーを作る操作はモデル20の場合異なってきますが、本書での解説を読み、ユーティリティプログラムを実行し画面に表示されるメッセージに従って行なえばよいでしょう。

また、モデル10の場合、ディスクユニットを起動したとき画面に表示される数字などは多少異なりますが、本書での操作には影響ありません。

要するに、本書で作るプログラムはいずれの機種でも何ら支障なく動きますのでご安心ください。

なお、本章での記事は、ナツメ社発行の『ＰＣ—8801mkII　プログラミング入門』と重複しますが、すでに理解している方は次章へお進みください。

1・2　システムディスクのコピーを作る

システムディスクは大切なものです。何らかの手違いでこわれることもありえます。万一の場合にそなえて、いまあなたがお持ちのシステムディスクとまったく同じものを作っておきましょう。

なお、この節では、２ドライブタイプのモデル30にもとづいて解説します。１ドライブのモデル20ご使用の方は、やり方が少し異なりますので、本節を読んでから、ユーティリティプログラムをロードし、実行してください。

また、本節および次節の記事は、姉妹編

『ＰＣ－8801mkII　プログラミング入門』

と重複する部分がありますが、これは、本書から入る方もいると考えたからですのでご諒承ください。上掲書を読んですでに理解している方は次のステップへお進みください。

システムディスクとは

本体購入時に標準添付されている２つ折の青いジャケットの中に、ディスクが２枚入っています。

このうち、ラベルに「N88DISK－BASIC　SYSTEM DISK」と書かれているのが、ここで扱うシステムディスクです。

もう一方はラベルに「GAME」と書いてありますね。これはゲームプログラムがおさめられたディスクです。こちらのほうは、みなさんが各自お楽しみください。

さて、システムディスクは、ディスクシステムを立ち上げ（起動させ）、ディスク操作（ディスクオペレーティング）を可能にする大切なディスクです。

このシステムディスクには、次のようなものがおさめられています。

❶ **システム**

ディスクを起動させる原動力となるもの。

❷ **ユーティリティプログラム**

ディスクにおさめられているプログラムとかデータを新しいディスクにそのままコピーしたりできる便利なプログラム群。

❸ **デモプログラム**

ＰＣ—8801mkⅡのコマンド、ステートメントをフルに活用したデモンストレーションプログラム。

❹ **チューター**

キーボードの使い方とかグラフィックスの考え方が学べる、いわば家庭教師的なプログラム。

❺ **各種データ**

デモンストレーションの画面作りに必要なグラフィックデータ。

概観すると、システムディスクの内容は上記のとおりです。

さて、私たちは、このシステムディスクと同じものをもう１つ作ろうというわけですが、次節でも使いますので、新しい「５インチ両面倍密度フロッピィディスク」を３枚用意してください。**〈注〉**

準備できたらスタートです。

〈注〉本書を学習するには新しいディスクが最低３枚必要です。しかし、ファイル管理を本格的にやろうと思うと、現行のプログラムとかデータをもう１つ余分にコピーしておく必要がありますので、できたら数枚購入しておくとよいでしょう。

システムの立ち上げ

この操作はもうわかりますね。

ディスプレイのスイッチを入れ、オリジナルのシステムディスクをドライブ１に入れます。そして本体の電源ＯＮです。

すると、アクセス表示用ＬＥＤが点灯し、使用するファイル数を問うメッセージが次のように表示されますから、「5」を入力してリターンキーを押しましょう。

```
How many files(0-15)? ■
```

```
Disk version [Aug 19,1983]
How many files(0-15)? 5
NEC N-88 BASIC Version 1.3
Copyright (C) 1981 by Microsoft
44581 Bytes free
Ok
■
```

なぜ「5」をキーインしたかといいますと、リターンキーを押すクセがついてしまいますと、ディスクで本格的にファイル操作をするようになったとき、エラーがでたりするからです。つまり、使用ファイル数を入力せずにリターンキーを押しますと、ファイル数は自動的に2に設定されます。しかし、ビジネス用のプログラムなどでは、ファイルを同時に3つ以上使うことはよくありますから、このような場合ファイル数が不足しエラーとなってしまいます。

ここでは「5」を入力しましたが、PC―8801シリーズではメモリ容量は十分ありますから、「10」くらい入力しておいてもよいと思います。

それはさておき、画面は前ページのようになりましたか？

これでN88DISK―BASICが立ち上がりました。

ファイルを画面に表示する

それでは、次のステートメントをキーインしてみてください。

キーインは大文字でも小文字でもかまいませんよ。

```
files 1 ↵
```

これは、**ドライブ1に入っているシステムディスクの中のファイルをすべて表示する命令です。**

「1」という数字がドライブ番号を示しています。ただし、ドライブ1の場合に限り、この数字を省略することができます。

もうおわかりと思いますが、**ドライブ2のフロッピィディスクのファイル名を見たい場合には、パラメータに「2」を指定**して

```
files 2 ↵
```

とキーインすればよいのです。「2」を省略すると、ドライブ1のディスクのファイルが表示されてしまいますから念のため……。

現在は、ドライブ2にはディスクは入っていませんから、これを実行すると、ピーッとブザー音がして、次のようなエラーメッセージが表示されます。ためしてみてください。

```
files 2
Disk offline
Ok
■
```

```
files 1
demo   .n88 7   tutor .n88 12 dskut1.n88 4   dskut2.n88 9   ngen   .n88 13
@load .n88 1   @exst *     2   demoex.n88 3   *      *    1   *demo0*     4
*demo1*     1   *demo2*     1   *tutor*     1   $pict0 dat 4   $pict1 dat 4
$pict2 dat 6   $pict3 dat 6   $pict4 dat 10 $pict5 dat 12 $pict6 dat 12
$pict7 dat 14 $pict8 dat 14
Ok
■
```

（$pict8 dat → ファイル名、14 → ファイルの大きさ）

ずいぶんいろいろなファイルがありますね。意味はよくわからないかもしれませんが、いまはかまいません。

上の画面では、ファイル名があって、そのうしろに8とか3などの数字が表示されていますが、これは、そのファイルがフロッピィディスク上でどのくらいの領域を使用しているかを示しています。単位はクラスタですが、頭のかたすみにおいておいてください。

ユーティリティプログラムのロード

上の画面の中に「dskut2. n88」というのがありますね。これが、システムディスクのコピーをしたりファイル転送をしたりするためのユーティリティプログラムです。「dskut」とは「Disk utility」の略ですね。

なお、「dskut」のうしろの数字が「2」とあるのは2ドライブのモデル30のための、「1」とあるのは1ドライブのモデル20のためのユーティリティプログラムです。

この「dskut2. n88」をメモリ上にロードして、これを実行することによりシステムディスクのコピーを作ります。

次のようにキーインしてください。ファイル名（プログラム名）は小文字で入力してください。

```
load "dskut2.n88"
```

（load → f・1 キーを押す、. → 小数点、" → ダブルクォーテーション）

キーインできましたらリターンキーを押してください。LEDが赤く点灯しロードが開始され、「Ok」が表示されれば終了です。

さあ、それではこのプログラムを実行しましょう。 f・5 キーを押してください。次のようなメニュー画面が表示されました。

```
DISK UTILITY for PC-8801MK2    dskut2

              M  E  N  U

        COPY DISK
        COPY DISK
        CREATE SYSTEM DISK [N88]
        CREATE SYSTEM DISK [ N ]
        CREATE DATA DISK
        TRANSFER FILES
        SET UP INFO        [N88]
        SET UP INFO        [ N ]

Select a feature and press RETURN key
```

表の中にプログラムメニューが7つありますね。そして現在、一番上の「COPY　DISK」が白い枠で囲まれて文字が反転しています。この白い枠はカーソル移動キー ↑ ↓ を使って希望するメニューのところへ移動することができます。ちょっとためしてみてください。どうですか？　移動したでしょう。

私たちがいま必要としているのは「COPY　DISK」プログラムですから、もう一度白い枠をこの上に重ね、リターンキーを押してください。

すると画面に次のようなメッセージが表示されます。意味は、「マスターディスクをドライブにセットしてください。セットしたドライブ番号は何番ですか？」ということです。

```
[[[ COPY DISK ]]]

Mount MASTER disk on drive.
Drive number? ■
```

私たちの場合にはシステムディスクがマスターディスクになります。ドライブ1にセットしてありますから、 [1] キーを押して ⏎ です。

```
1 ⏎
```

すると、新しいディスクをドライブにセットするよう指示してきます。そして、やはりドライブ番号をたずねています。

```
Mount NEW disk on drive
Drive number? ■
```

用意してある新しいディスクをドライブ2にセットしてリターンキーを押してください。

```
2 ⏎
```

今度は物理フォーマットをするかどうか聞いてきます。

```
Do you need physical formatting(y/n)? ■
```

物理フォーマットとは、市販されている5インチのフロッピィディスクはそのまま使うことができないので、使えるように初期化することをいいます。これ以上の説明は省略しますが、とにかく物理フォーマットという作業をしないと使用できない、と覚えておいてください。

ここでは次のようにキーインしましょう。

```
y ⏎
```

すると画面に次のような表示が現れ、物理フォーマット実行の確認を求めています。

```
Format a disk on drive 2
Sure(y/n)? ■
```

やはり「y」をキーインします。

```
y ⏎
```

「物理フォーマット中」のサインが次のように表示され、ディスクの初期化（イニシャライズ）が行なわれます。

```
Physical formatting.
```

しばらくして物理フォーマットが終了すると、いよいよマスターデ

ィスクに書き込まれている情報を新しいディスクへコピーする作業が開始されます。最後に次のような画面になってコピーが終了します。

```
Copying track 77
Copying track 78
Copying track 79

Completed.
Ok
■
```

さあ、これでシステムディスクを新しいディスクにコピーすることができたわけですが、うまくいったかどうか確かめてみましょう。

確認の方法はＦＩＬＥＳコマンドを使って、ドライブ１のディスクとドライブ２のディスクのファイル名を表示させてみればよいですね。

次のようにキーインしてください。

```
files 1 ⏎
```

```
files 2 ⏎
```

画面は次のようになりました。

```
files ◀──────────────────────────ドライブ1のディスクのファイル名を表示
demo  .n88 7  tutor .n88 12 dskut1.n88 4  dskut2.n88 9  ngen  .n88 13
@load .n88 1  @exst *    2  demoex.n88 3  *     *   1  *demo0*    4
*demo1*    1  *demo2*    1  *tutor*    1  $pict0 dat 4  $pict1 dat 4
$pict2 dat 6  $pict3 dat 6  $pict4 dat 10 $pict5 dat 12 $pict6 dat 12
$pict7 dat 14 $pict8 dat 14
Ok
files 2 ◀────────────────────────ドライブ2のディスクのファイル名を表示
demo  .n88 7  tutor .n88 12 dskut1.n88 4  dskut2.n88 9  ngen  .n88 13
@load .n88 1  @exst *    2  demoex.n88 3  *     *   1  *demo0*    4
*demo1*    1  *demo2*    1  *tutor*    1  $pict0 dat 4  $pict1 dat 4
$pict2 dat 6  $pict3 dat 6  $pict4 dat 10 $pict5 dat 12 $pict6 dat 12
$pict7 dat 14 $pict8 dat 14
Ok
■
```

どうですか？　ドライブ１のディスクの内容も、ドライブ２のディスクの内容もまったく同じですね。つまり、コピーに成功したのです。このコピー版のディスクには当然システムもコピーされていますから、これを使ってディスクを起動させることができるわけです。そこで、オリジナルのシステムディスクはどこか安全な場所に保管しておき、次節では、このコピーしたディスクを使うことにしましょう。

では、ドライブ１のシステムディスクを取り出し、ドライブ２のディスクをドライブ１にセットしてください。

1・3　システムだけをコピーする

12ページの画面を見てもわかるように、システムディスクにはいろいろなファイルが保存されていますが、この中には普段必要ないものがたくさんあります。グラフィックデータなどでずいぶんディスクのメモリを浪費しています。ですからここでは、ディスクを立ち上げるシステム（システムはファイルではありませんよ。ディスクユニットを起動させるための情報〈データ〉として、ディスクの所定の位置に書き込まれています）だけを新しいディスクにコピーすることにしましょう。

ユーティリティプログラムはまだメモリにありますから、これを、f・5 キーを押して実行しましょう。

それでは f・5 キーを押してください。すると、13ページのようにメニューが表示されますから、カーソル移動キーを押して白枠を上から２つ目のメニュー「CREATE　SYSTEM　DISK〔N88〕」のところへもっていってください。

そしてリターンキーを押します。画面は次のようになりました。

```
[[[ CREATE SYSTEM DISK ]]]

      N88 DISK-BASIC

Mount SYSTEM disk on drive.
Drive number? ■
```

システムディスクはドライブ１にセットしてありますから、

```
1 ↵
```

でよいですね。

すると次には、新しいディスクをドライブにセットするよう指示してきます。

```
Mount NEW disk on drive.
Drive number? ■
```

２枚目の新しいディスクをドライブ２にセットしてください。セットできたら 2 キーを押してリターンです。

前節と同じように物理フォーマットをするかどうかたずねてきます。

```
Do you need physical formatting(y/n)? ■
```

ここで「y」をキーインしリターンキーを押すと、

```
Format a disk on drive 2
Sure(y/n)? ■
```

のように確認を求めてきますから、ここでも「y」を入力しましょう。

```
y ⏎
```

すると、画面に次のメッセージが表示され、物理フォーマットが行なわれます。

```
Physical formatting.
```

続いてシステムフォーマットが行なわれます。

```
System formatting.
```

これは、システムデータを書き込めるようにイニシャライズしているのです。

そして最後に、システムだけをコピーし、次のような画面になって終了です。

```
Copying system.

Completed.
Ok
■
```

これで、システムだけをコピーする作業はすべて終りました。現在ドライブ2に入っている、システムだけをコピーしたフロッピィディスクを仮りに「ディスクA」としておきます。

ところで、もう1枚新しいディスクがありますね。

ドライブ2の「ディスクA」を取り出し、この新しいディスクをドライブ2にセットして、いま述べた方法で、やはりシステムだけをコ

ピーしてください。この節のはじめに戻って、 f·5 キーを押すところからやればよいのです。
さあ、実行してみてください。

どうですか？　うまくできましたか？
それでは、この、２つ目のシステムだけをコピーしたディスクを仮りに「ディスクＢ」としましょう。
以後、私たちは、ディスクＡとディスクＢを使うことになりますが、これらをセットするドライブは、ディスクＡをドライブ１に、ディスクＢをドライブ２にセットします。

ところで、いまはＡ、Ｂと名前は違いますが、内容はまったく同じものです。
これから先、ユーティリティプログラムはおおいに必要になりますので、ディスクＡにユーティリティプログラムだけコピーしておきましょう。
モデル20の方もいると思いますので、それを前提に手順を説明します。説明のとおりにキーインしてください。
ドライブ１にセットされている、コピーしたシステムディスクを取り出し、ディスクＡをドライブ１にセットしてください。ドライブ２のディスクＢは取り出してしまってけっこうです。
次のようにキーインしてください。

```
save "dskut2.n88"
```

"2" ← モデル20ご使用の方はこの数字「１」にする
save ← SHIFT キーと f·1 キーを押す

すると、いまメモリ上にあるユーティリティプログラムがディスクＡにセーブされます。
「Ｏｋ」が表示されれば、作業終了です。
これからは、この、ディスクＡをドライブ１に入れ、ディスクＢをドライブ２に入れるようにしましょう。ただし、モデル20の人は、ディスクＡに相当するディスクを続けて使用することになります。
なお、拡張機能（サウンド、タートルグラフィックなど）を使いたい人は、システムディスクの中の拡張機能パッケージプログラムをセーブしなければなりませんが、この方法は、

『ＰＣ—8801mkII　**プログラミング入門**』

の 102 ページをごらんください。

1・4　プログラムのセーブとロード

プログラムをせっかくキーインしても、電源を切ると、現在コンピュータの記憶領域（メモリ）にあるプログラムはすべて消えてしまいます。

ですから、これをどこかにたくわえておかなければなりません。その、たくわえる場所が小さな磁気盤、ディスクです。

この節では、サンプルとして小さなプログラムを１つ作り、これをディスクに保存し、それをもう一度メモリ上に呼びだす方法を学びましょう。

すでにごぞんじの方もいると思いますが、そのような方はここを飛ばして次へお進みください。

プログラムをディスクに保存することをセーブするといい、SAVE（セーブ）コマンドを使います。 SAVE

ここでは、プログラムをセーブし、それをもう一度メモリ上に取り出すことを確認すればよいので、サンプルプログラムは、ごく簡単なものにします。次のようにキーインしてください。

```
1 x=5
2 y=3
3 print x+y
```

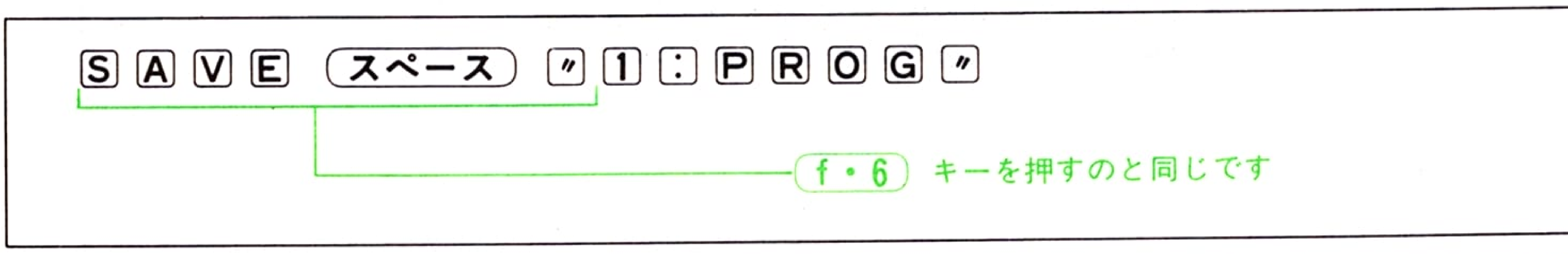
変数ＸおよびＹに、それぞれ５と３を代入し、その合計を画面に表示させるものです。

では、ホームクリアキーで画面をクリアし、リストコマンドで、プログラムがメモリに記憶されたかどうか確認しておきましょう。

```
list
1 X=5
2 Y=3
3 PRINT X+Y
Ok
■
```

プログラムをディスクにセーブする際、どんなプログラム名で保存しておくのか、プログラムに名前をつけなければなりません。この名前のことをファイル名といいますが、ここでは仮りに「prog」としておきます。 ファイル名

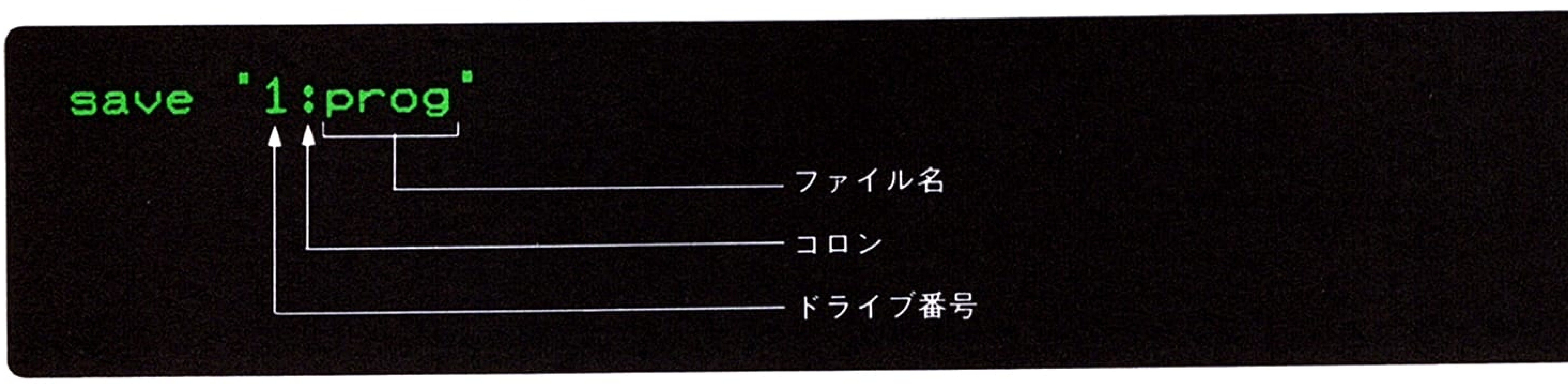
次のようにキーインします。まだ ⏎ キーは押さないでください。



「save ″ 」は (f・6) キーに定義されていますから、これを使えば楽ですね。

また、「prog」は、ここでは小文字にしてください。

画面は次のようになっていますか？



「save」は大文字でも小文字でもかまいませんよ。ただし、「prog」だけは小文字にしてください。**ファイル名の場合、コンピュータは大文字と小文字を区別します。**

ドライブ

〈注〉モデル20にはドライブ2はありません。

さて、私たちのコンピュータには、ディスクを入れる場所が2ついています。これを**ドライブ**といい、「1」という数字が書かれているほうを**ドライブ1**、もう一方を**ドライブ2**といいます。(注)

上の画面ではドライブ1を指定しました。つまり、「prog」をドライブ1側のディスクにセーブすることになります。

このように、SAVEコマンドの書式は、

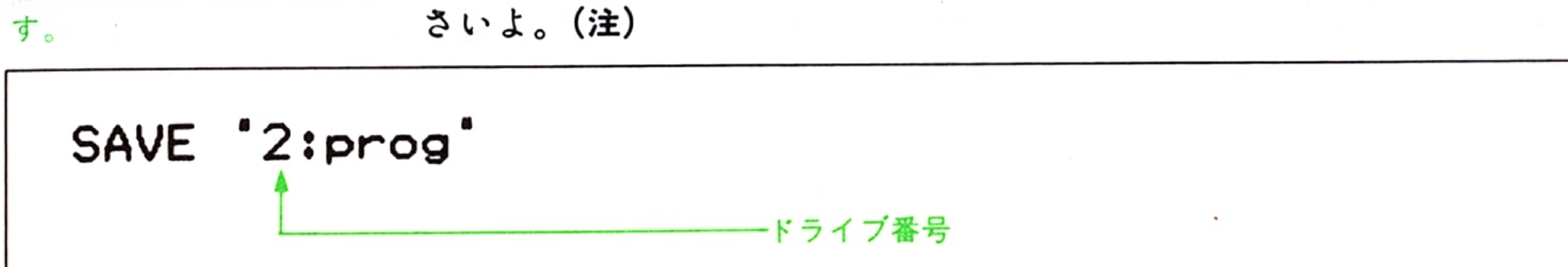
SAVE“〈ドライブ番号〉：〈ファイル名〉”

ファイル ディスプリプタ

〈注〉モデル20にはドライブ2がついていませんから「2：」は使うことができません。つまり、ドライブ番号はまったく不要になります。

で表わされ、ドライブ番号とファイル名をコロンで結んだものを**ファイルディスクリプタ**といいます。そして、**ドライブ1にセーブする場合には、ドライブ番号を省略できます。**つまり上の画面では、ファイルディスクリプタを、「1：」をはぶいて「prog」としてもいいのです。

それでは、いまディスクは入っていませんが、ドライブ2にセーブしたいときにはどうしたらよいでしょうか？　もうわかりますね。次のようにキーインすればよいのですね。これはキーインしないでくださいよ。(注)



さあ、おまたせしました。リターンキーを押してみましょう。
すると、赤いランプがつき、カチ、カチカチと音がして画面にセーブ終了の合図「Ok」が表示されました。

```
save "1:prog"
Ok
■
```

それでは、ほんとうにセーブされたかどうか確認してみたいと思います。次のようにキーインしてください。画面モードを変えます。

```
WIDTH 80 ⏎
```

画面がクリアされて、ホームポジションに「Ok」が表示されています。しかし、文字がずい分小さくなったでしょう。それに、ファンクションキー1から10までの定義内容がすべて表示されました。
次のステートメントをキーインしてください。

```
FILES ⏎
```

すると画面は次のようになりました。

```
files
dskut2.n88 9    prog    .    1
Ok
■
```

セーブされている

いかがですか？　いま私たちがセーブしたプログラムが見えますね。どうやら、うまくセーブできたようです。

アスキー形式のセーブ

これで、プログラムのセーブ方法を理解していただけたと思いますが、セーブにはもう1つの方法がありますので覚えておきましょう。
いま私たちが学んだセーブ方法は「非アスキー形式のセーブ」とい　非アスキー形式のセーブ

アスキー形式のセーブ

い、これから説明するのを「アスキー形式のセーブ」といいます。

たとえば「prog」をアスキー形式でセーブするときは、次のようにキーインします。

```
save "1:prog",a
```

（,a ── アスキー形式のセーブ）

前回のセーブ方法と異なるのは「,a」がついている点ですね。たったこれだけでよいのです。このように「,a」をつけることから、このセーブ方法を「Aオプションつきのセーブ」ともいいます。

なお、「a」は大文字でもかまいません。

アスキー形式と非アスキー形式の違いはディスクへの保存のされ方が違うのですが、いまは、前者のほうが後者より多少メモリを余計必要とする、くらいに考えておいてください。

ただ、アスキー形式のほうがディスク上のメモリを多少必要とするかわりにメリットもありますが、本書では非アスキー形式のセーブで話を進めます。

プログラムのロード

LOAD

フロッピィディスクにセーブされたプログラムをもう一度コンピュータのメモリに呼びだすことをロードするといい、LOAD（ロード）コマンドを使います。

その書式は、アスキー形式、非アスキー形式、いずれでセーブされたプログラムについても同じで、次のとおりです。

```
load "1:prog"
```

（"1:prog" ── ファイルディスクリプタ）
（load ── ファンクションキー1に定義されています）

つまり、セーブする場合の「SAVE」を「LOAD」に変えただけです。ファイルディスクリプタの表わし方も同じですね。ドライブ1にセーブしたプログラムはドライブ1から、ドライブ2にセーブしたプログラムはドライブ2からロードします。

それでは「prog」をさっそくロードしてみようと思いますが、本当にロードできたかどうか確認する意味で、コンピュータのメモリをクリアしてしまいましょう。

NEW

本体前面左のリセットボタンを押してコールドスタートしてもよいのですが、ここでは新しい命令を1つ覚えましょう。それは、NEW（ニュー）コマンドです。新しくする、新規にする、ということですね。

次のようにキーインしてください。

```
new
```

これで、メモリ内のプログラムはすべて消えてしまいました。ちなみに、(f・4) キーを押してプログラムリストを表示させてみてください。

```
new
Ok
list
Ok
■
```

どうですか？ プログラムリストは何も表示されませんね。つまり、メモリはすっかりクリアされてしまったのです。

それでは、前ページの書式にしたがってロードしてみてください。そして (f・4) キーでプログラムリストを表示させてください。

画面は次のようになりましたか？

```
load "1:prog"
Ok
list
1 X=5
2 Y=3
3 PRINT X+Y
Ok
■
```

ドライブ１のディスクからロードされたプログラム（1〜3行）

成功ですね。ではプログラム名を大文字にしてためしてみましょう。

```
load "1:PROG"
File not found
Ok
■
```

エラーメッセージ（File not found）

「ファイルが見つかりません」というエラーメッセージが返ってきました。ファイル名は、小文字でセーブしたら、小文字でロードしなければいけないのです。

いま画面は横80桁のモードになっています。40桁のほうがよいという人は、WIDTH文で変更してください。

1・5　ディスクの管理

ユーティリティプログラムの使い方、ご理解いただけましたか？最初はちょっととっつきにくいですが、慣れてくるとたいしたことはありません。ご安心ください。

まあ、あまり神経質にならないことです。

本書で使用するディスク

ところで本書では、1・3節で述べたように、システムとユーティリティプログラムをコピーしたディスクをドライブ1にセットしてプログラムファイル用に、システムだけをコピーしたディスクはドライブ2に入れてデータファイル用に使います。

ライトプロテクトシールとラベル

オリジナルのシステムディスクをごらんください。

ラベルが2枚貼ってありますね。1つは正方形、もう1つは、そのとなりにある長方形のラベルです。長方形のほうには、「N88DISK－BASIC　SYSTEM　DISK」とされています。

このシステムディスクには、右ページ下の図のように、右側に銀色のテープが貼ってあります。このテープのことをライトプロテクトシールといい、テープを貼った、ディスクの切り込みのことをライトプロテクトノッチといいます。

このシールを貼ることによって、ディスクに他の情報が書き込まれるのを防ぐことができます。

大切なディスク（いつも使うものはこの限りではありません）には必ず、このライトプロテクトシールを貼っておくことにしましょう。

また、ライトプロテクトシールと一緒に長方形のラベルもついていますから、あなたが使うディスクには、仕事名を書いて貼っておくと便利です。



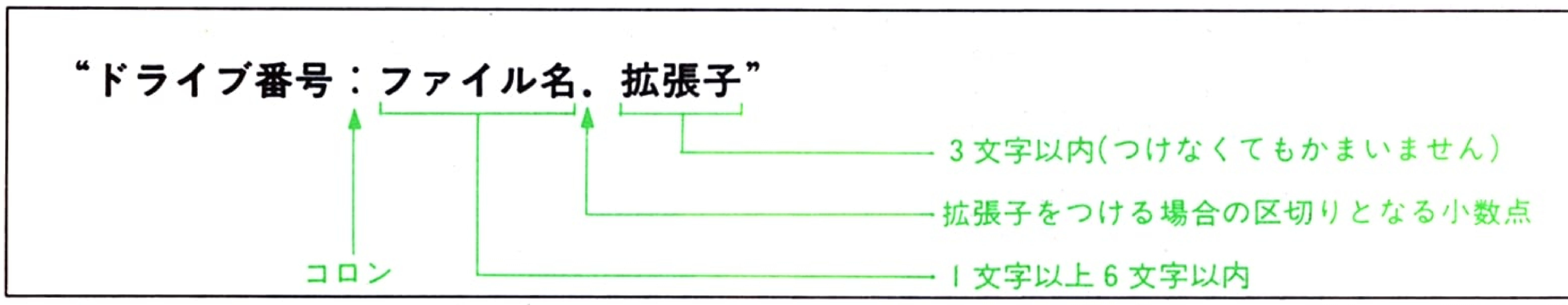
普段使うディスク



オリジナルのシステムディスク

1・6　ファイルディスクリプタについて

ここではファイル名のつけ方について、簡単に説明しておきましょう。



ドライブ番号は前に説明したように、ドライブ1の場合には省略することができますが、ドライブ2の場合には必ずドライブ番号を入れ、コロンで区切ります。

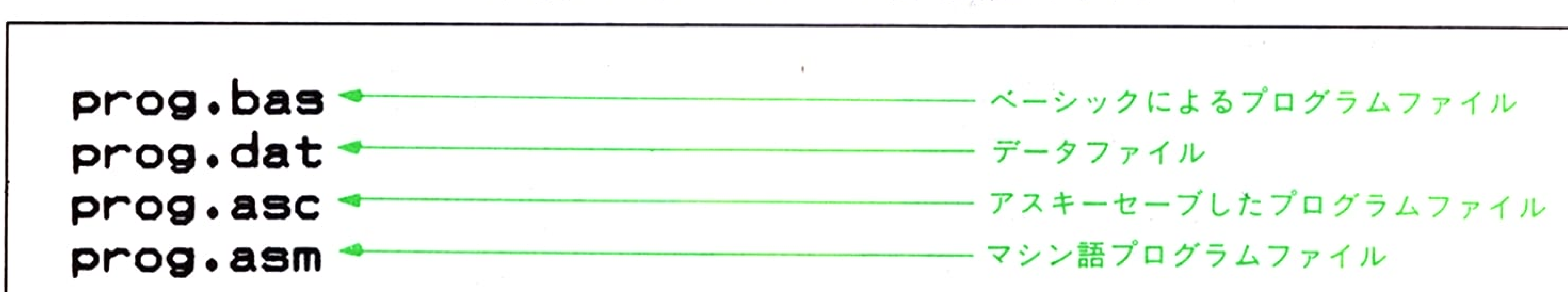
ファイル名は6文字以内で、英字、数字、カナ文字など、いずれも使用することができます。ただし、英字の大文字、小文字は別のものとして扱われますので注意してください。

次の小数点は、拡張子をつける場合に区切りとして使います。

最後の拡張子、これはなくてもかまいませんが、あとになってファイルを管理する場合大変役立ちます。

なお、ファイル名と拡張子を合わせたものをファイル名と考えてもよいのですが、ファイル管理上、区別しておいたほうが便利ですので、本書では分けて考えます。

拡張子のメリットについてですが、たとえば「prog」を例にとりますと、次のようにしておくと大変便利です。



「bas」はbasicの略、「dat」はdata の略、「asc」はascii の略、「asm」は assemble の略です。

なお、上の形でディスクにセーブしてFILESコマンドで画面に表示した場合、データファイルおよびアスキー形式のプログラムファイルには小数点は入りません。また、マシン語プログラムの場合には「＊」が小数点のかわりに表示されます。

1・7 ファイル名の変更と削除

何かのつごうでファイル名を変更したり削除したりしたい場合が生ずると思います。ここでは、その方法について簡単にふれておきます。

ファイル名の変更

ディスクに書き込まれているファイル名の変更は、次の書式に従って行ないます。NAMEコマンドを使います。 NAME

NAME 〈旧ファイルディスクリプタ〉 AS 〈新ファイルディスクリプタ〉

ファイルディスクリプタ中のドライブ番号は、新旧いずれも同じでなければいけません。

たとえば「prog」を「PROG」のように大文字に変更したい場合、次のようにキーインします。ディスクは仮りにドライブ1に入っているとしましょう。

```
name "prog" as "PROG"
```

これでファイル名が変更されます。ためしてみてください。そしてFILESコマンドで確認してください。

ファイル名の削除

プログラミングの過程で、サンプルプログラムとかテストプログラムなど、いろいろなプログラムがディスクにセーブされていきます。そんなとき、すでに不要になったものは、次のKILLコマンドでディスクから削除しておきましょう。 KILL

書式は次のとおりです。

KILL 〈ファイルディスクリプタ〉

たとえば、ドライブ1のディスクに入っている「prog」を削除したい場合、次のようにキーインします。

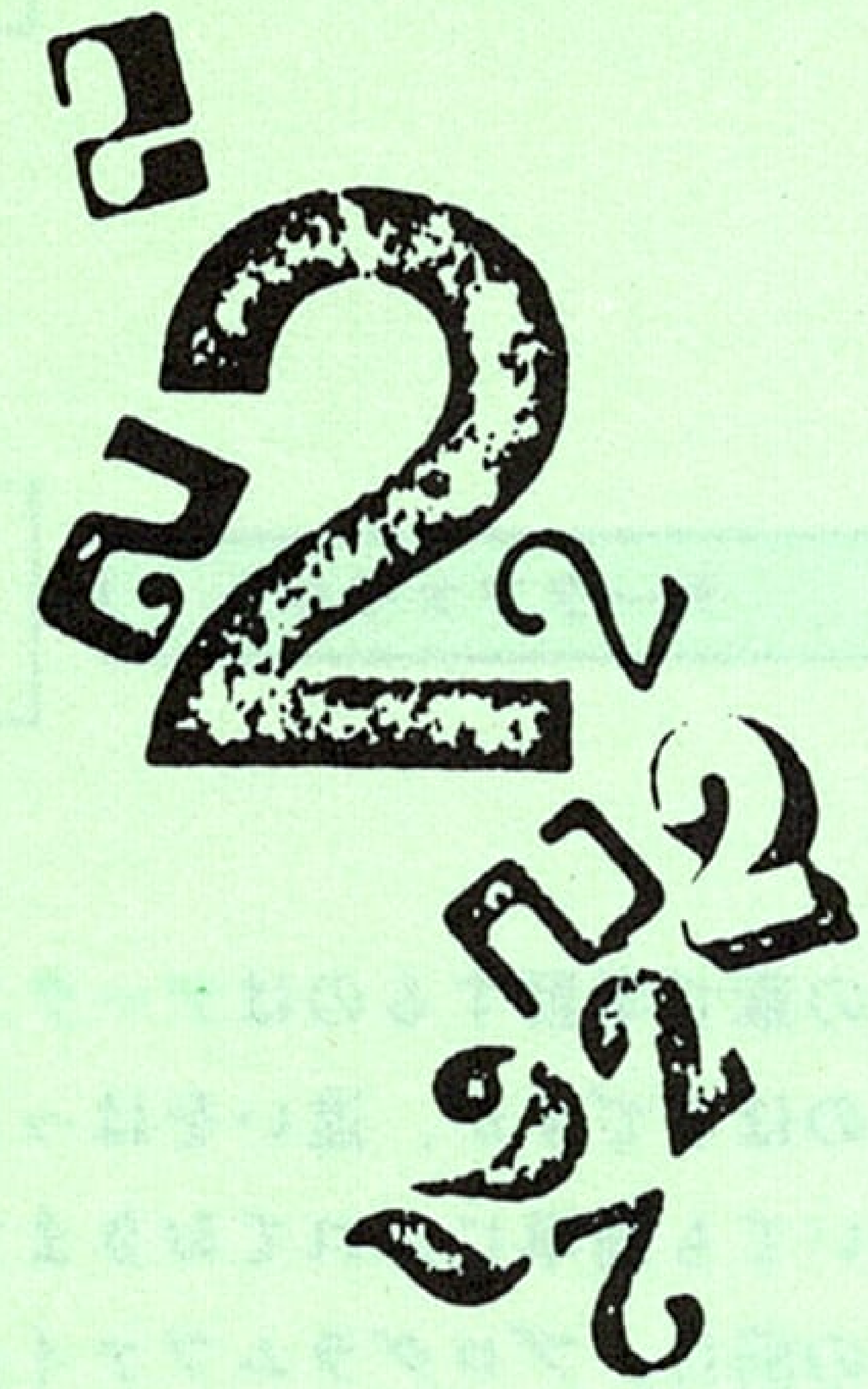
```
kill "prog"
```

ドライブ1ですから、ドライブ番号は省略してもかまいません。

間違えて、大切なプログラムを削除しないよう、くれぐれも注意してください。一度KILLすると復帰できませんので……。

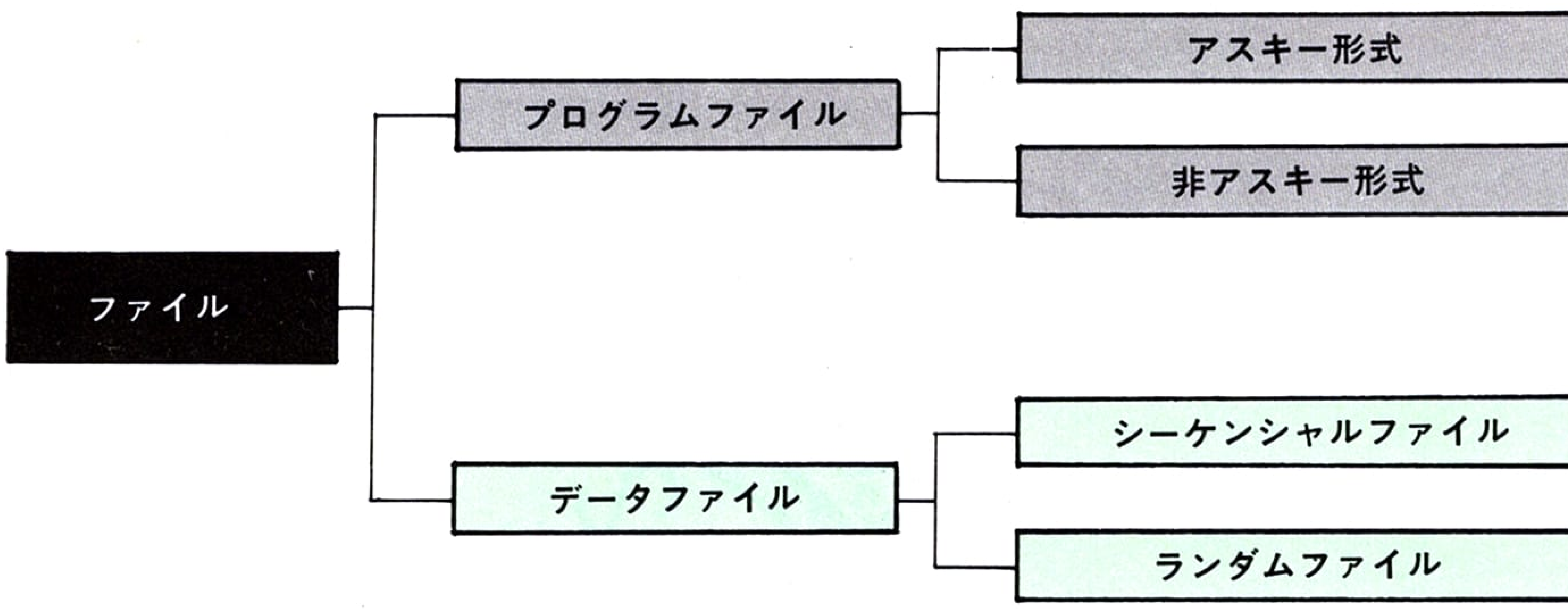
第2章

シーケンシャルファイル[I]



2·1　ファイルの種類

情報の集合体のことをファイルといいますが、これはその性質によって次のようにわけることができます。



この章で学習するのはデータファイルのうちのシーケンシャルファイルのほうですが、違いをはっきりさせるために、ランダムファイルについても簡単にふれておきます。

その前に、プログラムファイルとデータファイルの違いはもうおわかりですね。

プログラムファイル
データファイル

そうです、BASICプログラムなどをフロッピィディスクにセーブした場合にプログラムファイル、電話番号・住所・氏名といったデータをフロッピィディスクに書き込んだ場合にその集まりをデータファイルといいます。

シーケンシャルファイルもランダムファイルも同じデータファイルなのですが、フロッピィディスクに書き込まれる状態が違います。

「シーケンシャル」というのは「連続的な」という意味です。「ランダム」というのは直訳すれば「デタラメナ」ということですが、デタラメというよりは「連続的ではない」くらいに理解しておいたほうがよいようです。

シーケンシャルファイルとは

シーケンシャルファイルとは、フロッピィディスクに1番目のデータから連続的に書き込まれたファイルのことで、これをディスクから読み出そうとするときに、書き込まれた順序でしか読み出せません。

たとえば電話帳のファイルをシーケンシャルで作った場合、夏目さ

んという人の電話番号を知りたくても、もし50音順に書き込まれていれば、安藤さん、伊藤さん……と、夏目さんの前に登録してある人のデータを読み出していかなければなりません。

あなたは、カセットテープを使ってファイルを作りましたか？　あれがシーケンシャルです。

シーケンシャルファイルの特徴をまとめると次のとおりです。

シーケンシャルファイルの特徴

❶　データをロード・セーブする場合、必ずファイルの先頭から順次行います。

❷　ファイル内の一部だけを書き換えることができません。

❸　ファイルを変更できるのは、ファイルの最後に新しいデータを追加することだけです。

❹　プログラム面、あるいはファイル設計の面から見ても、ランダムファイルに比較して簡単です。

❺　先ほどの電話帳の例でいいますと、夏目さんの電話番号・住所・氏名を1レコード（このレコードの集まりがファイルです）と呼びますが、この大きさは255文字以内でしたら自由に変えられます。

レコード

意味があまりわからない人もいるかもしれませんが、「ランダムファイルより扱いが簡単だ」くらいに考えていただき読み進んでください。あとで実際にプログラムを組んでみればわかると思いますし、そのときになってもう一度読みなおしていただいてもけっこうです。

ランダムファイルとは

次にランダムファイルですが、やはり電話帳の例にそってお話しします。

電話帳ファイルをランダムファイルとしてフロッピィディスク上に作った場合、たとえば夏目さんにかかわるデータを知りたいとき、ほかの関係ないデータまで読み出す必要はなく、直接に取り出すことができます。したがって、時間的なロスもなく大変効率的です。

ランダムファイルについても特徴を列挙しておきましょう。

ランダムファイルの特徴

❶　ファイル内の各レコードを任意に読み書きできますので、処理時間を短縮できます。

❷　ファイル内の一部のデータだけを書き換えることができます。

❸　データの追加も自由です。

❹　シーケンシャルファイルに比べてプログラムが複雑になります。

❺　1つ1つのレコードの大きさは256バイト以内の一定バイト数に固定されています

2・1　ディスクの構造

この節ではN88DISK-BAIC で使用できるフロッピィディスクについて説明しますが、まだ実際にプログラムによるファイル操作をしていませんので少しわかりにくいかもしれません。とくに、フロッピィディスクの構造がどうなっているかということは、プログラミングする上であまり大切ではありません。ですから、ちょっととまどいを感じる人は、この節をとばして次へ進んでいただいてもかまいません。

N88DISK-BASICで使えるフロッピィディスクは次の3つです。

❶　8インチ標準フロッピィディスク（両面倍密度用）
❷　5インチミニフロッピィディスク（両面倍密度用）
❸　5インチミニフロッピィディスク（片面倍密度用）

あなたはどのフロッピィディスクをお持ちですか？

呼び名は多少違いますが、扱い方はどれも同じです。違うのは記憶容量が多いか少ないかということになります。

右のページをごらんください。構造を図解してあります。

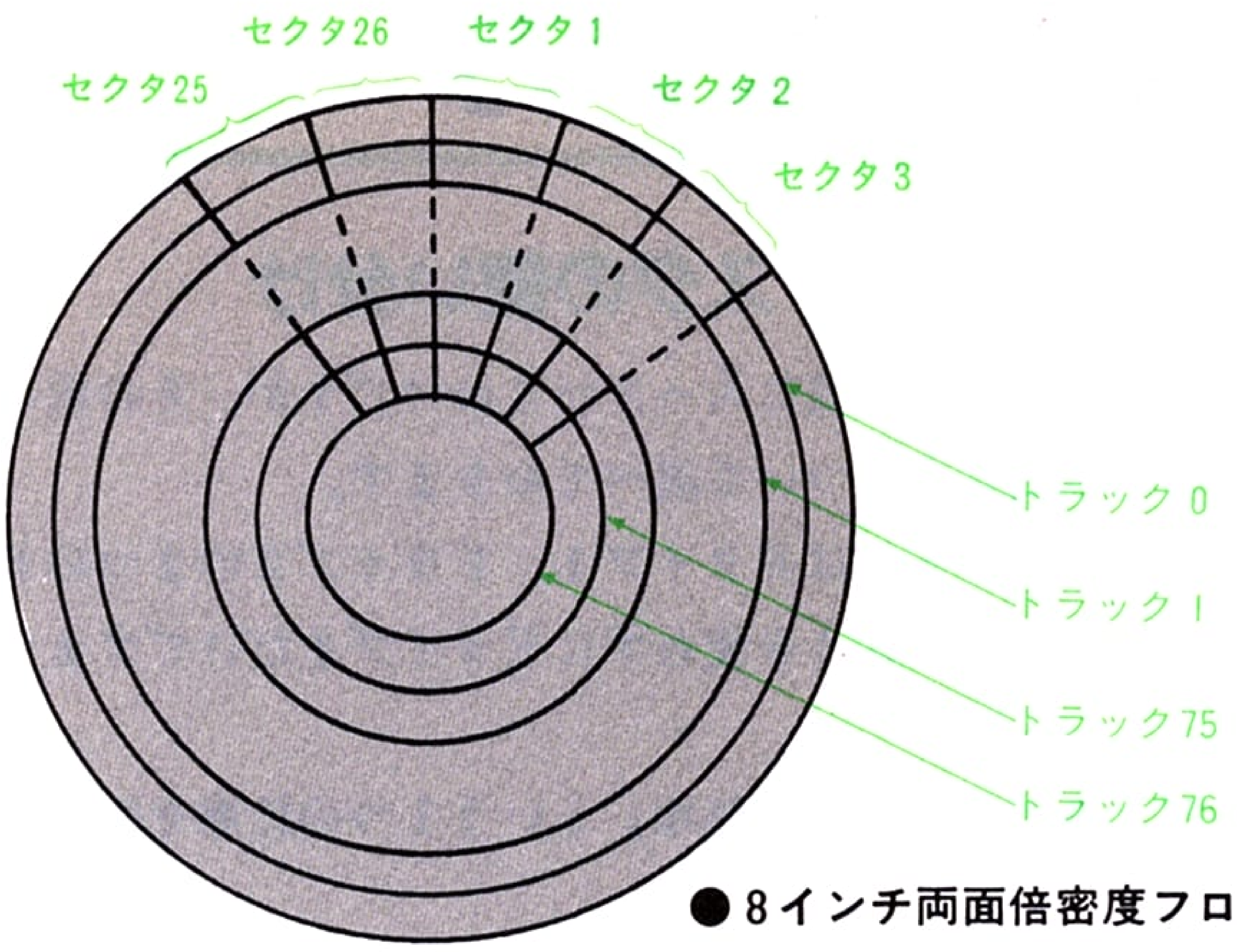
いずれも「セクタ」「トラック」「サーフェス」という単位で構成されていて、読み書きはセクタ単位で行われます。

1セクタは256バイト、文字に換算すると256字分ということになります。このセクタが図の円の1周分、8インチ両面倍密度ですと26セクタ、5インチのフロッピィディスクですと16セクタ集まったものをトラックと呼びます。サーフェスとは裏か表かということ、つまり、両面倍密度ですと裏表に読み書きできますが、片面倍密度では表しか読み書きできません。したがって、フロッピィディスクの種類によって総セクタ数が違ってきます。これらを表にまとめておきます。

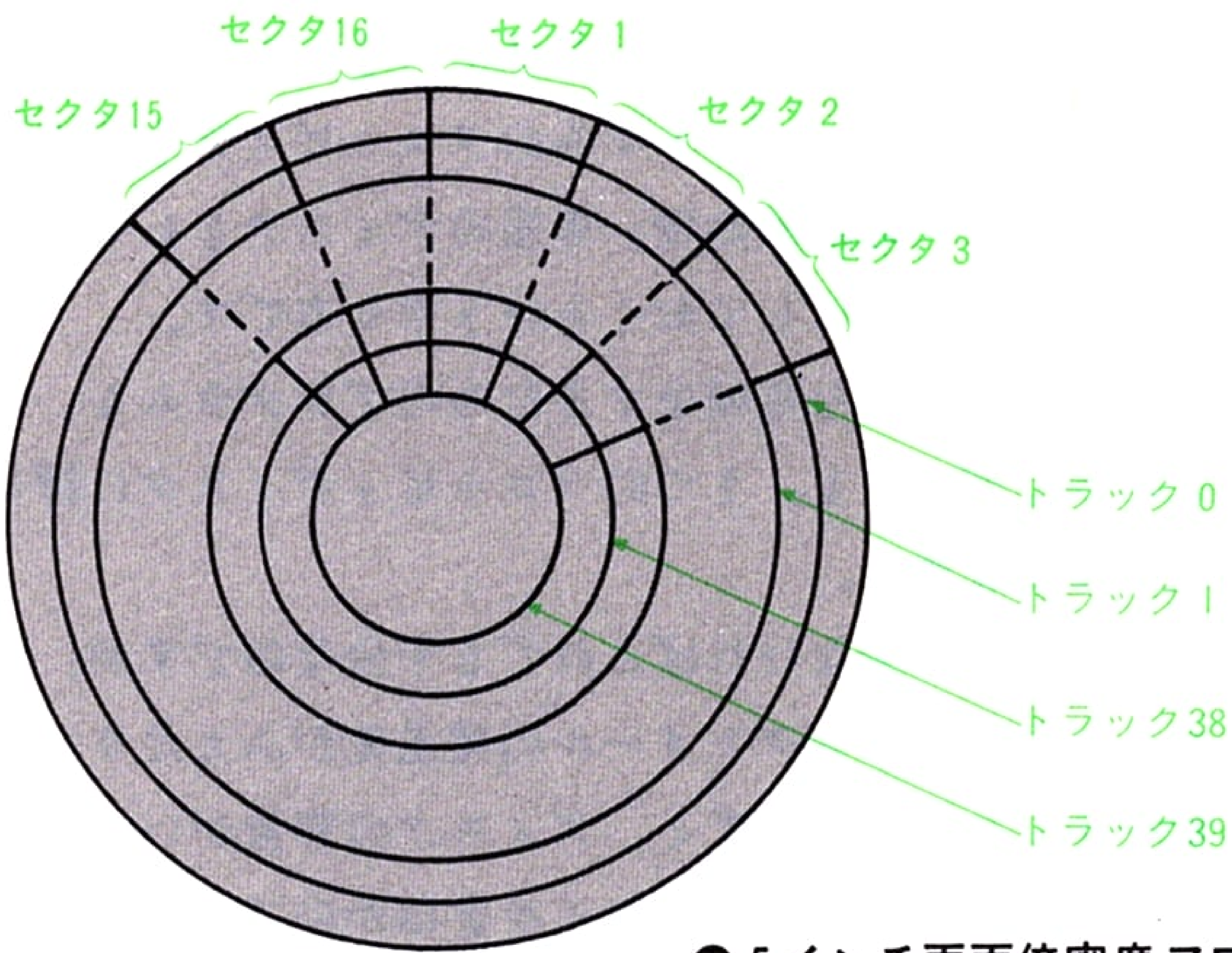
また、総セクタ数は次のように計算します。

総セクタ数＝1トラック当たりのセクタ数×トラック数×サーフェス

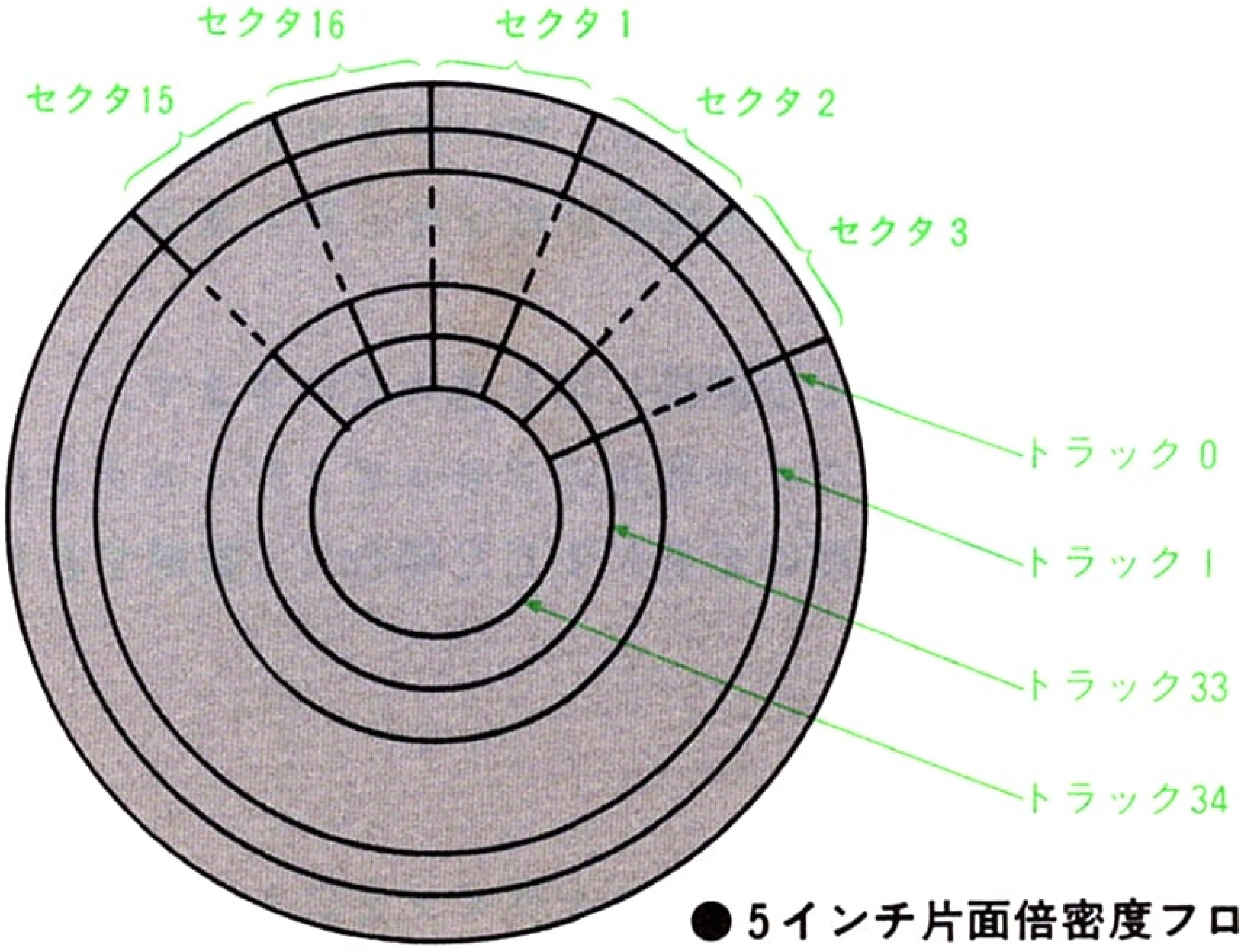
フロッピィディスク	サーフェス	トラック数	セクタ数	総セクタ数
8インチ両面倍密度	0か1か	77	26	4004
5インチ両面倍密度	0か1か	40	16	1280
5インチ片面倍密度	必要なし	35	16	660



●8インチ両面倍密度フロッピィディスク



●5インチ両面倍密度フロッピィディスク



●5インチ片面倍密度フロッピィディスク

2･3　ディスクの扱い方

フロッピィディスクの使い分け

これからいよいよ実践に入っていくわけですが、その前にフロッピィディスクの扱い方について述べておきます。

ディスクユニットにはドライブ1、ドライブ2の2つのドライブがありますが、ドライブ2を使わずにドライブ1だけですますこともできます。

ここでもやはり「電話帳処理」という1つの仕事を例にとってご説明します。

まず順序に従ってシステムディスクでN88DISK―BASICを起動させます。システムが起動したら、ドライブ1からシステムディスクを取り出し、すでにフォーマットずみのフロッピィディスクをドライブ1に入れ、これを「電話帳処理」専用のディスクとして使用し、ドライブ2は未使用のままにしておくというやり方です。

この場合、いろいろな処理をするためのプログラムも、電話番号とか氏名といったデータも、1つのフロッピィディスクに押し込んでおかなければなりません。

しかしデータ量が少なければこれでも十分なのですが、やがて得意先がふえるとか、ダイレクトメールの発送先がふえたりして保存するデータ量がふくらんできますと、ディスクの容量がいっぱいになり、プログラムファイルかデータファイルをほかのフロッピィディスクに移さなければならない事態も生じます。こうなるとプログラムも多少変更しなければならなくなりますし、とにかくめんどうです。

データファイル用
フロッピィディスク

プログラムファイル用
フロッピィディスク

このようなことを防ぐためにも、データファイル用のフロッピィディスクとプログラムファイル用のフロッピィディスクの2つを用意しておくとよいと思います。

この本でも、このことを前提に話を進めていきます。

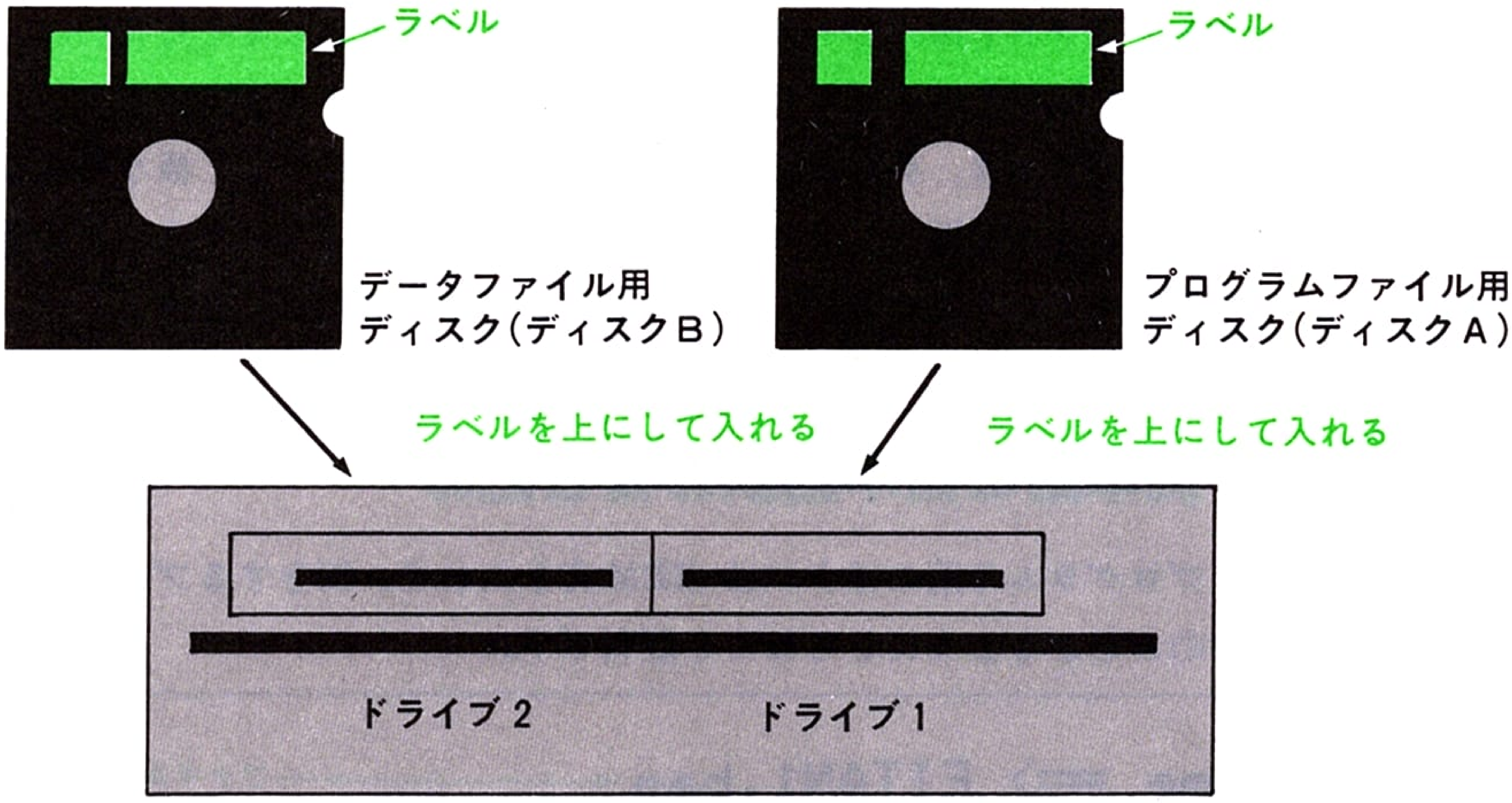
このようにデータファイル専用のフロッピィディスクを作っておけば、何かほかの事務処理の必要が生じた場合にも兼用できるというメリットもあります。

次にもう1つ。

データファイル用のディスクはドライブ2に入れ、プログラムファイル用のディスクはドライブ1に入れる習慣をつけるとよいと思います。これも、あなたがコンピュータによる事務管理に上達し、扱うデ

ータ量が多くなったときにおわかりいただけると思います。

私たちは1・3節で説明したように、システムおよびユーティリティプログラム「dskut 2 .n88」をコピーしたディスクAをドライブ1に、そしてディスクBをドライブ2に入れるようにしましょう。



ラベルを貼っておく

フロッピィディスクが1枚2枚のうちはいいですが、ふえてくるにつれて、何のためのディスクかはっきりさせておく必要があります。1・5節で説明したように、色分けされたラベルをディスクに貼り、「給与計算」「在庫管理」などの仕事名を書き込んでおくとよいでしょう。また、すでに貼ってあるラベルに書くときはディスクにキズがつかないようにサインペンなどを使ってください。

コピーをとっておく

この作業も忘れてはならないことの1つです。

せっかく苦心して作り上げたデータファイル、あるいはプログラムファイルの入ったディスクをどんなことで壊してしまうかもしれません。少しめんどうでも、プログラムを修正したり、データを追加したりしたときには必ずユーティリティプログラム「dskut 2 .n88」を走らせ、コピーをとっておくようにしてください。ユーティリティプログラムをロードしてから、コピー用ディスクをドライブ2に入れ、マスターディスクをドライブ1に入れてプログラムをRUNさせればよいのです。コピーしたディスクは大切に安全な場所に保管しておき、変更のつど取り出してコピーしなおすようにしましょう。

2・4　ディスクにデータを書き込む

それでは、これからいよいよデータの読み書きを学んでいきます。
フォーマットずみのディスクをドライブ2にセットしてください。

ここで取り上げるプログラムは「英単語練習プログラム」です。しかしこの本では、英語の勉強が目的ではありませんから、扱う単語もやさしい基本単語にしておき、データの扱い方、プログラミングの基本をマスターしていただくことに重点をおいてお話しします。

英単語と日本語訳の書き込み

REM文

プログラムの前のほうには REM 文を入れて、何をするプログラムなのかわかるようにしておくと便利です。

```
10 ' file name ==> EITAN1.bas      ←ファイル名
20 ' 1984.1.25                     ←作成年月日
30 ' sequential data write         ←仕事名
40 ' save "EITAN1.bas"
50 '
```

●本書では、画面モードは目の健康を考慮して、40桁×20行あるいは40桁×25行を採用しています。左右80桁のほうが好きだという方は、プログラムの初期設定を変更してみてください。この場合、LOCATE文も修正しないと画面のバランスがくずれます。

行番号40のステートメントは、プログラムを手直ししたのちセーブするとき便利です。

つまり、カーソルを「save」の s の上に重ね、(DEL) キーを5回たたくと s が行頭に移動しますから、ここでリターンキーを押せば、このプログラムをセーブすることができます。

次に、画面の行数と文字の大きさですが、入力にはそれほどたくさんの字数は必要ありませんので左右40桁、行数は20行としました。

また、ファンクションキー表示も消しておきましょう。

```
100 CONSOLE 0,20,0,0:WIDTH 40,20   ←ファンクションキー無表示(3番目の0)
110 CLS                            ←画面をクリア
```

シーケンシャルファイルに数値とか文字列を書き込むとき、カセットテープの場合には PRINT ＃－1 を使いましたが、ディスクバージョンでは次の書式によります。

```
WRITE #（ファイル),（式）
```

「式」には数値式あるいは文字式が入ります。

また「ファイル番号」には0から15までの整数を指定できますが、システムディスクを起動させたときに画面に表示された

ファイル番号

```
How many files(0-15)?
```

というメッセージがありましたが、このとき入力した数値をこえてはいけません。「同時にいくつファイルを使いますか？」というあなたへの問いかけでしたね。

それではさっそく書き込み部分のプログラムを作ってみましょう。次のようになります。

```
120 INPUT "ｴｲｺﾞ          ";A$     ← 英単語と日本語の入力
130 INPUT "       ﾆﾎﾝｺﾞ  ";B$
140 WRITE #1,A$,B$               ← ディスクへデータを書き込む
```

さあ、RUNさせてみてください。適当な英単語と日本語訳を入力してみましょう。

おや？　ピーッというブザー音とともにエラーメッセージが返ってきました。精神衛生上よくない音ですが、まあ我慢しましょう。

次のような画面になってコマンド待ちになってしまいました。

```
ｴｲｺﾞ ? airplane          ← どちらも入力後リターンキーを押す
     ﾆﾎﾝｺﾞ? ﾋｺｰｷ
File not open in 140     ← エラーメッセージ!!
Ok
█
```

直訳すると「140行でファイルがOPENされていない」ということです。ファイルが開かれていないとはどういうことでしょうか。

実はファイルというのは、たとえてみればノートのようなもので、開いてなければ書き込みができないのです。

あなたがノートに何か書くとき、開いてから字を書きますね。ディスクの読み書きも原理はこれとまったく同じなのです。

ここで再び、シーケンシャルファイルとランダムファイルの違いについてふれておきます。

フロッピィディスクをノートにたとえますと、シーケンシャルファイルの場合、1ページ目から順序よく書いていかなければならないのです。一方、ランダムファイルの場合は、自分の好きなページに、つ

シーケンシャルファイルとランダムファイルの違い

まり、10ページ目でも50ページ目でも自由に書き込むことができるのです。

また、すでに書き込んであるノートのあるページの文章だけ読みたくなったとき、シーケンシャルファイルですと、自分の意志とは関係なく、1ページ目から、1、2、3、……と順序よくページをめくっていかなければならないのです。

ところが、ランダムファイルの場合、ノートにインデックスがついているようなもので、直接自分の希望するページを開いて読むことができるのです。ずいぶん便利ですね。

これで2つのファイルの違いがイメージ的に以前よりはっきりしてきたのではないかと思います。

ファイルのOPENとCLOSE

さて、画面にエラーメッセージが出たままストップしてしまいましたが、いまのたとえ話でもおわかりいただけたように、ファイルという名のノートをまずOPENする、つまり開かなくてはいけないのです。そして、これも大切なことですが、書き終えたら閉じておく、つまりCLOSEしなければならないのです。OPEN文は常にCLOSE文と対で使われます。書式は次のとおりです。

OPEN する

CLOSE する

```
OPEN "ファイル名" FOR ( モード ) AS #(ファイル番号)

CLOSE #(ファイル番号),#(ファイル番号),・・・・・・・
```

ファイル名は、そのファイルがドライブ1にあるのでしたらドライブ番号は省略できますが、ドライブ2にあるファイルに対しては次のようにドライブ番号をつけるのでした。

2:EITAN1.dat

OPEN文のファイル番号は0から15までの値を指定することができますが、前ページで説明したように、起動時に指定したファイル数をこえてはいけません。

次にモードですが、シーケンシャルファイルの場合、アクセスの方法によって3種類あります。

モード

```
OPEN "2:EITAN1.dat" FOR OUTPUT AS #1 ←アウトプットモード
OPEN "2:EITAN1.dat" FOR INPUT AS #1 ←インプットモード
OPEN "2:EITAN1.dat" FOR APPEND AS #1 ←アペンドモード
```

アウトプットモードはデータをディスクに書き込むときに、インプットモードはディスクに書き込まれているデータを読み出すときに、そして最後のアペンドモードは、すでにディスクにセットされているシーケンシャルファイルの末尾にデータを追加したいときに使用します。現在私たちが必要としているのはデータ書き込みですからアウトプットモードということになります。ステートメントは次のようになります。

アウトプットモード
インプットモード
アペンドモード

〈注〉1ドライブのディスクユニットでは、ファイルディスクリプタの中のドライブ番号は省略してください。以後、いちいち注意書きをそえませんが同じ扱いです。

```
115 OPEN "2:EITAN1.dat" FOR OUTPUT AS #1
150 CLOSE #1
160 END
```

ファイル書き込みのモードでOPENする（115行）
書き込みが終ったらファイルを閉じる（150行）

さあ今度はエラーが出ずに実行できるでしょうか。RUNコマンドで走らせてみてください。

```
ｴｲｺﾞ ? airplane
      ﾆﾎﾝｺﾞ? ﾋｺｰｷ
Ok
■
```

確かに成功したようですが、1単語登録しただけで終ってしまいました。もう少しデータ数をふやすことはできないでしょうか。

120行のステートメントにメッセージを入れ、さらに「99」を入力したらEND処理へ飛び、その他の場合はデータを書き込んだのちに120行へいくようにしましょう。次のステートメントを追加します。

```
120 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
125 IF A$="99" THEN GOTO 150
145 GOTO 120
```

```
ｴｲｺﾞ (ｵﾜﾘ=99)? airplane
      ﾆﾎﾝｺﾞ? ﾋｺｰｷ
ｴｲｺﾞ (ｵﾜﾘ=99)? apple
      ﾆﾎﾝｺﾞ? ﾘﾝｺﾞ
ｴｲｺﾞ (ｵﾜﾘ=99)? bee
      ﾆﾎﾝｺﾞ? ﾐﾂﾊﾞﾁ
ｴｲｺﾞ (ｵﾜﾘ=99)? bicycle
      ﾆﾎﾝｺﾞ? ｼﾞﾃﾝｼｬ
ｴｲｺﾞ (ｵﾜﾘ=99)? ■
```

実行させた状態は前ページのとおりです。ずいぶん単語を登録できるようになりましたが、入力したデータが上から下まで表示されて、少々見ばえが悪いようです。もうひと工夫して、単語を5つ入力したら画面をクリアするようにしましょう。LIST コマンドでプログラムを画面に表示してみましょう。

```
10 ' file name ==> EITAN1.bas
20 ' 1984.1.25
30 ' sequential data write
40 ' save "EITAN1.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 CLS
115 OPEN "2:EITAN1.dat" FOR OUTPUT AS #1
120 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
125 IF A$="99" THEN GOTO 200
130 INPUT "      ﾆﾎﾝｺﾞ";B$
140 WRITE #1,A$,B$
145 GOTO 120
150 CLOSE #1
160 END
```

カウンタ

行番号140と145の間にカウンタを入れて、カウンタが5になったら画面をクリアするようにします。このときカウンタを0にもどしておくのを忘れないでください。

```
142 C=C+1
143 IF C=5 THEN C=0:CLS
```

RENUM

これで完成です。最後に行番号100以降がふぞろいですのでリナンバリングして整理しておきましょう。

```
renum 100,100
```

このプログラムをセーブしてから RUN させて、次の単語を入力してデータファイルを作ってみてください。

```
airplane・・・・・・・・ﾋｺｰｷ          apple・・・・・・・・・・ﾘﾝｺﾞ
bee・・・・・・・・・・・・・・ﾐﾂﾊﾞﾁ      bicycle・・・・・・・・ｼﾞﾃﾝｼｬ
blackboard・・・・・ｺｸﾊﾞﾝ          castle・・・・・・・・・ｵｼﾛ
city・・・・・・・・・・・・・ﾄｼ         college・・・・・・・・ﾀﾞｲｶﾞｸ
comic book・・・・・ﾏﾝｶﾞ           desk・・・・・・・・・・・ﾂｸｴ
```

2·5 ディスクのデータを読む

この節では、前節でフロッピィディスクに書いたデータを読んで、それをディスプレイに表示してみます。実際に書き込みに成功したかどうかまだ不安です。はやく確かめたいものですね。

データを読むときも先ほどのノートの考え方と同じです。つまり、ノートを開いて（OPENして）、読み終ったら閉じる（CLOSEする）のです。ただしこの場合、モードが読み出しのモード（インプットモード）になります。形は次のとおりです。

インプットモード

```
OPEN "2:EITAN1.dat" FOR INPUT AS #1
```

（INPUT ── インプットモード）

このステートメントの最後に＃1というファイル番号がついていますが、これについて補足説明しておきます。

ＰＣ—8801mkII がディスクの読み書きをする場合、実は直接行うのではなく本体内のバッファエリアという場所を中継して行うのです。このバッファエリアの数はディスク起動時の

バッファエリア

```
How many files(0-15)?
```

で指定した数だけ確保されます。

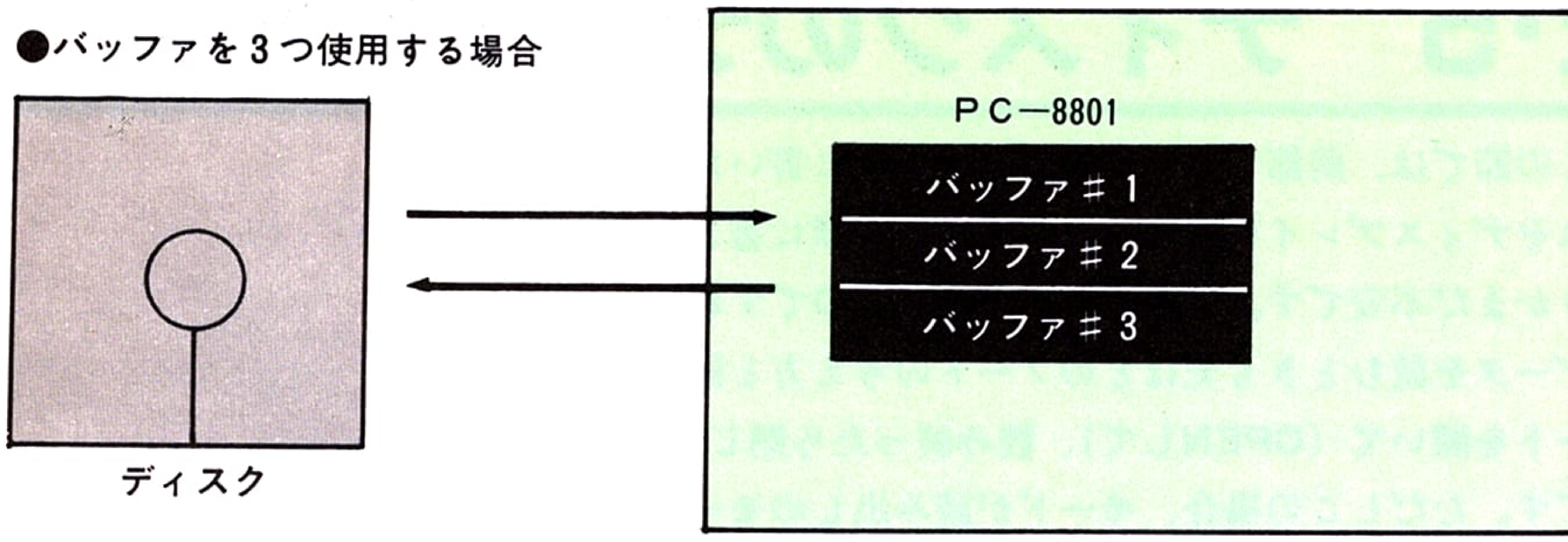
上のメッセージでは(０－15)となっていますから16のバッファエリアを使うことが可能ですが、実際のプログラムでファイルを同時にこんなにたくさんOPENすることもありませんし、また１つのバッファで256バイト必要ですから、メモリもそれだけよけいにとられてしまいます。

普通は５くらいを入力しておいて、この範囲内でうまくやりくりするほうがよいでしょう。

たとえば、１つのプログラムで６つのファイル（仮にＡ、Ｂ、Ｃ、Ｄ、Ｅ、Ｆとしておきます）を使うとしても、＃１のバッファエリアをＡに、＃２のバッファエリアをＢに、＃３のバッファエリアをＣに、というようにいつも限定して割り当ててしまうのではなく、たとえば、＃２のバッファエリアを使っているファイルＢをCLOSEすれば、ここを残りのＡ、Ｃ、Ｄ、Ｅ、Ｆは自由にOPENして使うことができるのです。つまり、５つのバッファエリアを同時に６つ以上のファイルの入出力で使いさえしなければ、どう使おうと自由なのです。

プログラムの流れの中で、使ってないファイルはCLOSEして、バッファエリアをほかのファイルに貸してあげればよいでしょう。

●バッファを３つ使用する場合



また、どんなにデータ量がぼう大であっても、１つのバッファには256バイトの広さしかありませんから、これを単位として読み書きを行っています。

話しをもとに戻しましょう。

インプットモードでファイルをOPENしましたら、次はデータを読まなければなりませんが、これにはINPUT＃文を使い、その書式ははは次のようになります。

INPUT＃

```
INPUT #1,A$,B$
```

INPUT＃文のうしろに続く変数は、数値変数であれ文字変数であれ、コンマで区切ればよいのです。

このプログラム名を「EITAN2.bas」として、フロッピィディスクからデータを読むプログラムを作りますと次のようになります。

```
10 ' file name ==> EITAN2.bas
20 ' 1984.1.25
30 ' sequential data read
40 ' save "EITAN2.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 CLS
120 OPEN "2:EITAN1.dat" FOR INPUT AS #1
130 INPUT #1,A$,B$
140 PRINT A$,B$
150 GOTO 130
160 CLOSE #1
170 END
```

プログラム名（10行）
セーブ用 REM 文（40行）
データを読む（130行）
ディスプレイに表示（140行）
CLOSE文（160行）

前節で書き込んだデータを読んでディスプレイに表示するようにしてあります。40ページのデータは書き込んでありますか？

それではRUNしてみてください。

おやおや、またピーッというブザー音とともにOkの表示がでてコマンド待ちとなりました。

```
airplane       ヒコーキ
apple          リンゴ
bee            ミツバチ
bicycle        ジテンシャ
blackboard     コクバン
castle         オシロ
city           トシ
college        ダイガク
comic book     マンガ
desk           ツクエ
Input past end in 130
Ok
█
```

ファイルに書かれていたデータ

エラーメッセージ

このエラーメッセージ、

```
Input past end in 130
```

は、ディスクを扱うようになると、初めの頃よく出会うメッセージです。意味は「行番号130で(ファイルの)終りを過ぎているのにまだ(データを)読もうとしている(INPUTしようとしている)」といったところです。

プログラムを見てください。

130行でデータを読んで140行で画面に表示し、150行で再び読み込み処理へ飛ばしています。これではファイルにデータがなくなってもまだ読み込もうとしてしまうはずです。どこかでファイルの末尾をチェックさせて、このループから抜け出さなければなりません。

EOF関数

この、データチェックのための関数がEOF(End of fileの略)です。コンピュータは、この関数の値が−1(真、つまりファイルの末尾であるということ)か0(偽、つまりまだファイルがあるということ)かによって判断しているのです。

ですから、この関数をデータを読むステートメントの前に入れなければなりません。

```
125 IF EOF(1) THEN 160
```

EOF(1)、つまり＃1のファイルがもしEnd of fileならば、ファイルのCLOSEをします。そして、行番号150のGOTO文を変更して、125行へ飛ばすようにします。

```
150 GOTO 125
```

プログラムを実行してみましょう。今度はエラーもなく無事に単語を画面に表示できたと思います。

しかしこれでは、ディスクからデータを読んで、それをディスプレイに表示したというだけで、英単語を表示し、それに対して答えるという形式がととのっていません。少し手直ししましょう。

140行では英単語だけを表示し、その意味を声を出して答え、答え終ったら ⏎ キーを押し、続いて正解を表示するようにステートメントの変更と追加をします。

```
140 PRINT A$
145 INPUT "  ニホンゴ" ヲ ド"ウゾ" (OK=RET)";R$
147 PRINT "  コタエ ==> ";B$
```

さあ、キーインできましたか？

できたらRUNしてみてください。

```
airplane
  ニホンゴ" ヲ ド"ウゾ" (OK=RET)?
  コタエ ==> ヒコーキ
apple
  ニホンゴ" ヲ ド"ウゾ" (OK=RET)?
  コタエ ==> リンゴ"
bee
  ニホンゴ" ヲ ド"ウゾ" (OK=RET)?
  コタエ ==> ミツバ"チ
bicycle
  ニホンゴ" ヲ ド"ウゾ" (OK=RET)? ■
```

←答えたらリターンキーを押す

これで、データを読み出すという1つの仕事を処理するプログラムとしては完成なのですが、問題と正解が画面をスクロールするだけではちょっと感じがでません。1問答えたら画面を消去し、次の出題というふうにしてみましょう。

また、表示位置もLOCATE文を使って調整してみました。

次に変更・追加ステートメントを示します。

```
140 LOCATE 7,5:PRINT A$
145 LOCATE 10,6:INPUT "  ﾆﾎﾝｺﾞ ｦ ﾄﾞｳｿﾞ (
OK=RET)";R$
147 LOCATE 10,8:PRINT "  ｺﾀｴ ==> ";B$
148 FOR I=1 TO 3000:NEXT I ←──時間つぶし
149 CLS ←──画面をクリアして次の問題に
```

行番号148のFOR～NEXT文は時間つぶしの常とう手段です。このステートメントを入れておかないと、すぐ次に画面消去のステートメントがありますので、行番号147のコタエはほとんど表示されないように見えます。

時間つぶし

〈注〉ステートメントが長くなる場合、上の145行のように40桁で折り返すか、コロン以下で改行したりしますが、みなさんは続けて打ち込んでください。

ちょっと退屈な勉強をしたのですから、労をねぎらう意味で、「ゴクロウサマデシタ」となぐさめの言葉を入れておきましょう。

```
165 LOCATE 10,7:PRINT "ｺﾞｸﾛｳｻﾏ ﾃﾞｼﾀ"
166 FOR I=1 TO 5000:NEXT I
```

そしてリナンバリングとセーブをして完了です。

```
renum 130,125 ↵
save "EITAN2.bas" ↵
```

```
10 ' file name ==> EITAN2.bas
20 ' 1984.1.25
30 ' sequential data read
40 ' save "EITAN2.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 CLS
120 OPEN "2:EITAN1.dat" FOR INPUT AS #1
130 IF EOF(1) THEN 210
140 INPUT #1,A$,B$
150 LOCATE 7,5:PRINT A$
160 LOCATE 10,6:INPUT "  ﾆﾎﾝｺﾞ ｦ ﾄﾞｳｿﾞ (
OK=RET)";R$
170 LOCATE 10,8:PRINT "  ｺﾀｴ ==> ";B$
180 FOR I=1 TO 3000:NEXT I
190 CLS
200 GOTO 130
210 CLOSE #1
220 LOCATE 10,7:PRINT "ｺﾞｸﾛｳｻﾏ ﾃﾞｼﾀ"
230 FOR I=1 TO 5000:NEXT I
240 END
```

2・6　データを追加する

これまで私たちは、ディスクにデータを書き込み、それを読み出して画面に表示するところまで学習してきました。

データ量が多くて、とてもひと息に入力できないとか、何かの都合で次回にまわしたいとか、いろいろ事情も生じてくると思います。

残りのデータを現在作成されているデータファイルに後日あらためて追加して書き込みたいという場合はどうしたらよいでしょうか。

OUTPUTモードではデータ書き込みはできましたが、このモードでデータファイルをOPENすると、すでに書かれているデータがご破産になり、あらためて最初から入力することになります。つまり、苦心して書き込んだデータが消えてなくなってしまうのです。

ファイルポインタ

これはメモリの中にファイルポインタというものがあって、ファイルをOUTPUTモードでOPENすると常にこのポインタがファイルの先頭を指し示し(セットされる)、新規に書き込みを行うことになるからです。初めのうちは、このことを知らずにデータを消失してしまうこともありますので、注意してください。このようなミスを防ぐ方法は、のちほど説明します。

INPUTモードでもファイルポインタはファイルの先頭にセットされますが、読み出しモードですので心配ありません。

データをシーケンシャルファイルに追加して書き込みたい場合は、

APPEND

APPENDモードを使います。

```
OPEN "2:EITAN1.dat" FOR APPEND AS #1
```

(APPEND ← データ追加のモード)

すでにフロッピィディスクの中にある「EITAN1.dat」というデータファイルを#1のバッファを使ってAPPENDモードでOPENしなさい、という意味です。

そして、このOPEN文が実行されると、ファイルポインタはファイルの末尾にセットされ、追加して書き込みできる準備が完了したことになります。

APPENDモードはOUTPUTモードと同じように、データを書き込むという点においては変わりありませんから、その他のステートメントは同じで、OUTPUT用のプログラムをスクリーンエディタで多少変更するくらいで使うことができます。

それでは書き込み用のプログラムファイル「EITAN1.bas」をロードして、スクリーンエディタで修正してみましょう。

```
load "EITAN1.bas" ⏎
```

ロードできましたらLISTコマンドでディスプレイに表示してみてください。

```
list
10 ' file name ==> EITAN1.bas
20 ' 1984.1.25
30 ' sequential data write
40 ' save "EITAN1.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 CLS
120 OPEN "2:EITAN1.dat" FOR OUTPUT AS #1
130 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
140 IF A$="99" THEN GOTO 200
150 INPUT          ﾆﾎﾝｺﾞ";B$
160 WRITE #1,A$,B$
170 C=C+1
180 IF C=5 THEN C=0:CLS
190 GOTO 130
200 CLOSE #1
210 END
Ok
```

（20行目：これはあなたの作成年月日に）

上のプログラムリストの中で、スミアミになっているところだけ修正すればよいのです。プログラム名は「EITAN3.bas」とします。

次のようにしてください。

```
10 ' file name ==> EITAN3.bas

30 ' sequential data append

40 ' save "EITAN3.bas"

120 OPEN "2:EITAN1.dat" FOR APPEND AS #1
```

修正が終わりましたら、とりあえず**セーブ**しておきましょう。行番号40のステートメントを使うと便利です。

カーソルを行番号40のsaveのs の上に重ね、DEL キーを5回たたいて ⏎ です。キーインする手間がはぶけますね。

```
40 ' save "EITAN3.bas"
```

↑ここにカーソルを重ね(DEL)キーを5回たたいて⏎

セーブできました。これで安心です。

それではさっそくためしてみましょう。次の基本単語を追加することにします。(カナ)キーの切り換えを正確にしてください。

```
friend·······トモダ゛チ      iris··········アヤメ
kindergarten·ヨウチエン      lily··········ユリ
map··········チズ゛          pear··········ナシ
```

実行結果は、次のような画面になって終りです。

```
エイゴ゛ (オワリ=99)? pear
         ニホンゴ゛? ナシ
エイゴ゛ (オワリ=99)? 99
Ok
█
```

画面に Ok がでています。しかし、これはシーケンシャルファイルへのデータの追加が確実にできたことの保証ではありません。パソコンは処理が終わればいつも Ok サインをだすのですから……。まだまだ安心できません。前節で作った読み出しのプログラムで確かめてみましょう。

ここで1つ新しい知識を。いままではプログラムを実行するのに、まずロードして(f・5)キーを押していましたね。これを一挙にやってしまう楽な方法があります。2つ示しますから、どちらの方法でもかまいません、キーイン後⏎を押してみてください。

Rオプション付のロード

```
❶ load "EITAN2.bas",r ⏎

❷ run "EITAN2.bas" ⏎
```

❶も❷も機能は同じです。最終画面は pear が消え「ゴクロウサマ」が表示されて終りになります。

```
pear
         ニホンゴ゛ ヲ ド゛ウゾ゛ (OK=RET)?

         コタエ ==> ナシ
```

データ追加のプログラムを実行するとおわかりのように、画面に突然データ入力をうながすメッセージがでてきて、ちょっととまどいを感じる方もいらっしゃるかもしれません。もう少し余裕をもたせてみましょう。要するに、もう少しパソコンと取り組んでいるという感じを持たせたいものです。

1つは、しばし画面に「データ　ノ　ツイカ　ヲ　シマス」というメッセージを入れること。

もう1つは、最終登録単語と登録語数を表示し、それを確認してから追加処理に入ることです。

ここで「EITAN3.bas」をロードしてください。

画面構成は次のようにしたいと思います。

```
            ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ

        ﾏﾂﾋﾞ ﾉ ﾀﾝｺﾞ ==> pear
        ﾄｳﾛｸ ｿｳｽｳ  ==> 16 words

              OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ? ■
```

これら2つの処理は、画面消去をしてからAPPENDモードでファイルをOPENするまでの間、つまり行番号110と120の間で行わなければなりません。しかし、この処理ルーチンは少し行数もふえそうですし、現在のスペースではたりそうもありません。RENUMコマンドで行番120を140行にしてスペースをとっておきましょう。

```
renum 140,120 ↵
```

このくらいアキがあれば間に合いそうです。

まず、次のステートメントをキーインします。

```
112 LOCATE 10,5:PRINT "ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ"
```

次に、データファイルの末尾にある単語と、登録してある語数を変数に代入しなければなりません。

これはINPUTモードでファイルをOPENしてデータを読み飛ばし、ディスプレイに表示しないようにすればよいと思います。末尾の単語は当然読み出し用の文字変数に残りますから、これを画面に表示すればよいのです。また、読み出しルーチンでカウンタを入れておけば、登録してある単語の総数も得られますね。

さっそくコーディングしてみましょう。

ディスクのデータを読む部分は次のとおりです。前節で学習ずみですので、すぐ理解できると思います。

```
114 OPEN "2:EITAN1.dat" FOR INPUT AS #1
115 IF EOF(1) THEN GOTO 119   ← データがなければCLOSE
116 INPUT #1,A$,B$
117 C=C+1                     ← カウンタ
118 GOTO 115
119 CLOSE #1
```

ごらんのように、行番号116でデータを読んでいますが、画面に表示はしていません。A＄、B＄には、前のデータが追い出されて新しいデータが代入されるというように、先頭のデータから末尾のデータへと仕事がくりかえされ、**最終的には一番最後のデータが残る**ことになります。

カウンタ

Cはカウンタで、英単語と日本語を1組としてカウントしています。

次に画面表示部分のプログラムを示します。

```
120 PRINT                                   ← 改行
121 LOCATE 7,7:PRINT "ﾏﾂﾋﾞ ﾉ ﾀﾝｺﾞ==> ";A$
122 LOCATE 7,8:PRINT "ﾄｳﾛｸ ｿｳｽｳ   ==>";C;"words"
123 PRINT                                   ← 改行
124 LOCATE 12,10:INPUT "OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ";R$
125 CLS                                     ← 画面クリア
```

（122行の「==>"」の位置を指して）Cは数値変数ですから符号分も出力され、ここにスペースを入れると英単語とアタマゾロエになりません

これでほぼ完成ですが、もう1つ大切なことがあります。それは、**このあとのメイン処理でカウンタにCを使っていますので、これをゼロクリアしておかなければいけない**ことです。

```
126 C=0
```

そして総仕上げ、リナンバリングです。

行番号がだいぶふぞろいですので、きちんとつけなおしておきましょう。

```
renum 120,112
```

行番号112を行番号120とし、10きざみにリナンバリングするというわけです。

プログラムリストを次に示します。

```
10 '  file name ==> EITAN3.bas
20 '  1984.1.25
30 '  sequential data append
40 '  save "EITAN3.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 CLS
120 LOCATE 10,5:PRINT "ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ"
130 OPEN "2:EITAN1.dat" FOR INPUT AS #1
140 IF EOF(1) THEN GOTO 180
150 INPUT #1,A$,B$
160 C=C+1
170 GOTO 140
180 CLOSE #1
190 PRINT
200 LOCATE 7,7:PRINT "ﾏﾂﾋﾞ ﾉ ﾀﾝｺﾞ==> ";A$
210 LOCATE 7,8:PRINT "ﾄｳﾛｸ ｿｳｽｳ  ==>";C;"words"
220 PRINT
230 LOCATE 12,10:INPUT "OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ";R$
240 CLS
250 C=0
260 OPEN "2:EITAN1.dat" FOR APPEND AS #1
270 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
280 IF A$="99" THEN GOTO 340
290 INPUT "      ﾆﾎﾝｺﾞ";B$
300 WRITE #1,A$,B$
310 C=C+1
320 IF C=5 THEN C=0:CLS
330 GOTO 270
340 CLOSE #1
350 END
```

●ストップキーをうっかりして押してしまったとか、シンタックスエラーでプログラムが止まってしまったときは、必ず「CLOSE」をダイレクトモードで入力し、リターンキーを押してください。

次の単語を入力してテストしてみてください。

私たちは、OUTPUTモードからはじめて、ちょうど20語登録したことになります。

poster・・・・・ﾎﾟｽﾀｰ　　seat・・・・・・・・ｻﾞｾｷ
student・・・・ｶﾞｸｾｲ　　wolf・・・・・・・・ｵｵｶﾐ

2·7 データ修正ルーチン

これでシーケンシャルファイルへのデータ書き込み・追加、そして読み出すこともできるようになりました。

しかし、これでもまだ不備な点があるのです。それは何かと申しますと、OUTPUTモード、APPENDモードでの書き込みのとき、画面では5単語を1組にして表示し、終われば次の画面へというようにしておりましたが、単語は英語・日本語をペアにしてその都度書き込んでいました。つまり、キーインしたスペルが正しかったかどうかの確認をしていません。

前にもお話ししましたが、シーケンシャルファイルでは一度書き込まれたものはあとで修正するというわけにはまいりません(注)。ですから、なおいっそう慎重を期したいものです。苦労して打ち込んだデータの中にミスがあったら、あまりいい気分はしませんね。

〈注〉まるきり修正できないというわけではありませんが、一度配列に読み込み、その中から修正データをサーチし、修正後ふたたび書き込みを行なわなければならないので手間がかかります。

そこで本節では、修正するかどうかのルーチンを作って確認をうながすようにプログラム「EITAN3.bas」を手直ししてみましょう。

「EITAN3.bas」のセーブは忘れていませんか?

セーブしてから読み進んでください。

DELETE

まず、「EITAN3.bas」の行番号250から350までは作りなおさなければなりませんので、削除をします。DELETEコマンドを使えばいいですね。

```
delete 250-350 ⏎
```
← 行番号250から350までを削除

行番号10、20、40を修正してリストをとると次のようになります。

```
10 ' file name ==> EITAN4.bas
20 ' 1984.1.25 ｼｭｳｾｲ ﾙｰﾁﾝ ｦ ﾂｲｶ
30 ' sequential data append
40 ' save "EITAN4.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 CLS
120 LOCATE 10,5:PRINT "ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ"
130 OPEN "2:EITAN1.dat" FOR INPUT AS #1
140 IF EOF(1) THEN GOTO 180
150 INPUT #1,A$,B$
160 C=C+1
```
(行10の注記: ← 新しいファイル名)
(行40の注記: ← 新しいファイル名)

```
170 GOTO 140
180 CLOSE #1
190 PRINT
200 LOCATE 7,7:PRINT "ﾏﾂﾋﾞ ﾉ ﾀﾝｺﾞ==> ";A$
210 LOCATE 7,8:PRINT "ﾄｳﾛｸ ｿｳｽｳ  ==>";C;"words"
220 PRINT
230 LOCATE 12,10:INPUT "OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ";R$
240 CLS
```

行番号120以降はサブルーチンにします。 **サブルーチン**

まず、次のステートメントをキーインしてください。

```
115 '
116 '/// last data read ///   ←サブルーチンの注釈
117 *DAT.READ   ←サブルーチンのラベル
```

サブルーチンは、ラベルだけでも十分ですが、さらに具体的な注釈を入れておくと便利です。また、サブルーチンには必ずRETURN文をつけなければいけませんでした。

```
250 RETURN
```

これでサブルーチンとしての形ができあがりました。RENUMを使って、このサブルーチンを行番号1000からにしましょう。

```
renum 1000,115
```

リナンバリングを実行したことによって、行番号1000以降は次のようになりました。途中は省いて前後だけ示してあります。

```
1000 '
1010 '/// last data read ///
1020 *DAT.READ
1030 LOCATE 10,5:PRINT "ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ"
1040 OPEN "2:EITAN1.dat" FOR INPUT AS #1


1140 LOCATE 12,10:INPUT "OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ";R$
1150 CLS
1160 RETURN
```

次にデータ入力の部分を作ります。プログラムは次のようになります。

```
120 GOSUB *DAT.READ
130 DIM AA$(5),BB$(5)
140 OPEN "2:EITAN1.dat" FOR APPEND AS #1
150 FOR I=1 TO 5
160 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";AA$(I)
165 IF AA$(I)="99" AND I=1 THEN GOTO 210
170 IF AA$(I)="99" AND I<>1 THEN N=I-1
    :GOSUB *DAT.CORRECT:GOTO 210
180 INPUT "      ﾆﾎﾝｺﾞ";BB$(I)
190 NEXT I:N=I-1
200 GOSUB *DAT.CORRECT:GOTO 150
210 CLOSE :END
```

まず最初に行番号120で、ファイルを読んで末尾のデータを画面に表示するサブルーチンへ飛びます。これは前節のケースとまったく同じです。

行番号130で配列を宣言していますが、これは、入力した英語と日本語をそれぞれ記憶し、修正ルーチンで画面に表示させるために使うものです。このプログラムでは5つの単語を1セットにして考えていますので、10個以下ですから配列宣言の必要はないのですが、あとあとのことを考えますと、習慣づけておいたほうがよいでしょう。

FOR～NEXT

行番号150と190の間で5組ごとの単語入力のFOR～NEXTループを作っています。

ここで気をつけなければいけないのは、行番号190でループから抜け出すとき、Iをマイナス1してNに代入していることです。

次のようなダイレクトステートメントを実行してみてください。

```
for i=1 to 10 : next i : print i
 11
Ok
■
```

ちょっと考えると、この例の場合FOR～NEXTループから抜け出たとき、Iの値は10と思いがちですが、実際には11が入ってしまいます。Iに10が入っていると感違いをしてIをNに代入して、プログラムのどこかで、FOR J=1 TO Nなどとしますと、Jは11回ループすることになりますから、へんなことになってしまいます。

FOR～NEXTループを抜け出たとき、あるいは終りのマーク「99」が入力されたとき、データ修正のサブルーチンへ飛ばします。

データ修正のサブルーチンは次のとおりです。

```
2000 '
2010 '/// data correct sub ///
2020 *DAT.CORRECT
2030 CLS
2040 LOCATE 5,1:PRINT "ﾆｭｳﾘｮｸ ﾃﾞｰﾀ ｲﾁﾗﾝ"
2050 FOR J=1 TO N                          ┐
2060 LOCATE 2,2+2*J:PRINT AA$(J)           ├─英単語と日本語を表示
2070 LOCATE 10,3+2*J:PRINT BB$(J)          │
2080 NEXT J                                ┘
2090 LOCATE 7,2
2100 INPUT "ｼｭｳｾｲ ｼﾏｽｶ  ( y/n )";ANS$
2110 IF ANS$="n" OR ANS$="N" THEN GOTO 2180
2120 LOCATE 5,1:PRINT "ｼｭｳｾｲ ｶﾞ ﾅｹﾚﾊﾞ ｶｰｿﾙ ｦ"
2130 LOCATE 5,2:PRINT "ﾀﾝｺﾞﾉ ﾐｷﾞﾄﾞﾅﾘﾍｲﾄﾞｳｼﾃ RET"
2140 FOR J=1 TO N                          ┐
2150 LOCATE 0,2+2*J:INPUT "";AA$(J)        ├─カーソルを移動
2160 LOCATE 8,3+2*J:INPUT "";BB$(J)        │
2170 NEXT J                                ┘
2180 FOR J=1 TO N                          ┐
2190 WRITE #1,AA$(J),BB$(J)                ├─ディスクへの書き込み
2200 NEXT J                                ┘
2210 CLS
2220 RETURN
```

行番号2030－2040　画面をクリアして見出しを入れます。

行番号2050－2080　行番号160と180のINPUT文で配列にしまっておいた値を表示します。表示位置はLOCATE文によって指定します。ループ変数Jの値が1から5まででではなくNまでとなっていますが、これは途中で「99」が入力されて強制的にループから抜け出る場合があるからです。

「99」が入力されたとき表示する値は、そのときのIの値の1つ前まででよいわけですから、このサブルーチンに飛んでくる前にマイナス1した値をNに代入しています。

また行番号150－190のFOR～NEXTループを5回くりかえしたときも、先ほどの説明のように、N＝I－1としてからこのサブルーチンにこなければなりません。

行番号2090－2110　修正するかどうかの確認です。修正がなければ、2180行へ飛びディスクにデータを書き込みます。

行番号2120－2170　修正する場合、画面にメッセージを表示します。

修正方法

修正のしかたは、カーソルが1つめの英単語の先頭にセットされますから、なおしたければ正しいスペルを入力して ⏎ キーを、修正の必要がなければ ➡ キーを使ってその単語の末尾の文字の右隣までカーソルを移動させて ⏎ キーを押します。

いずれにしても次にカーソルは日本語の先頭にセットされますから、同じ操作をくりかえせばよいわけです。

行番号2140から2170のFOR－NEXTループ処理を終わると、行番号2180からのデータの書き込みをし、画面をクリアして新たに単語入力ルーチンへ戻ります。

この節で扱っている英単語追加登録のプログラムの場合、もう少し簡単な修正ルーチンですましてもよいのですが、売上げ伝票とか給料明細などを扱うようになるとどうしても必要なことですし、またプログラミングの勉強にもなりますので採用することにしました。画面レイアウトなどは、みなさんが目的・好みに応じてそれぞれ工夫してみてください。

さあ、忘れずにプログラムをセーブしておきましょう。セーブはプログラムをなおすたびに、こまめに行う習慣をつけたほうがよいと思います。先を急いでついうっかりセーブし忘れて悲惨な目にあうこともけっこうあるものです。

```
save "EITAN4.bas" ⏎
```

プログラムの実行

それでは f･5 キーを押してプログラムを実行してみましょう。最初に次の初期画面となります。

```
      デ゛ータ ノ ツイカ ヲ シマス

   マツヒ゛ ノ タンコ゛==> wolf
   トウロク ソウスウ  ==> 20 words

         OK ナラ RETキー ヲ ト゛ウソ゛? ■
```

⏎ キーを押すと単語入力モードになりますから、次のように5つの単語を入力してみましょう。beach を間違えて beech と入力してしまいました。これは修正ルーチンでなおすことにします。

```
ｴｲｺﾞ (ｵﾜﾘ=99)? station
     ﾆﾎﾝｺﾞ? ｴｷ
ｴｲｺﾞ (ｵﾜﾘ=99)? beech
     ﾆﾎﾝｺﾞ? ﾊﾏﾍﾞ
ｴｲｺﾞ (ｵﾜﾘ=99)? ship
     ﾆﾎﾝｺﾞ? ﾌﾈ
ｴｲｺﾞ (ｵﾜﾘ=99)? tiger
     ﾆﾎﾝｺﾞ? ﾄﾗ
ｴｲｺﾞ (ｵﾜﾘ=99)? bird
     ﾆﾎﾝｺﾞ? █
```

aをeと入力してしまった

bird に対する日本語「トリ」を入力し ⏎ キーを押すと画面は次のようになります。

beech を修正しなければなりませんから当然「y」を入力します。

```
      ﾆｭｳﾘｮｸ ﾃﾞｰﾀ ｲﾁﾗﾝ
        ｼｭｳｾｲ ｼﾏｽｶ  ( y/n )? y

   station
          ｴｷ
   beech
          ﾊﾏﾍﾞ
   ship
          ﾌﾈ
   tiger
          ﾄﾗ
   bird
          ﾄﾘ
```

次は実際に修正をする画面です。

```
シュウセイ ガ゛ ナケレバ゛ カーソル ヲ
タンゴ゛ノ ミギ゛ト゛ナリヘイト゛ウシテ RET

? station■
          エキ
  beech
          ハマベ゛
  ship
          フネ
  tiger
          トラ
  bird
          トリ
```

最初のカーソル位置

修正がなければ、この位置まで移動してリターンキーを押す

はじめにカーソルがstationのsの上にセットされますから、修正がなければ ➡ キーを使ってnの右隣りの位置までカーソルを移動し、↵ キーを押します。すると次にカーソルは「エキ」のエの上に重ねられますから、同じ操作をしてください。

beechのb の上にカーソルがきましたら、正しいスペルを入力して ↵ キーを押します。これで修正されたことになります。

「トリ」のところまで同じ操作をくりかえしますと、画面はクリアされて単語の入力画面になります。終りなら「99」を入力しましょう。

```
エイゴ゛ (オワリ=99)? 99
Ok
■
```

さあ、うまくいったでしょうか。データ読み出しのプログラムを走らせて確認してみてください。

```
run "EITAN2.bas" ↵
```

2·8 メニュープログラムを作ろう

以上、「英単語練習プログラム」を例にとってシーケンシャルファイルについて説明してきました。

アウトプットモード、インプットモード、アペンドモード、いかがでしたか？　思ったほどむずかしくはなかったと思います。

これまでに私たちがドライブ1のフロッピィディスクにセーブしたプログラムファイルは次の4つです。

```
"EITAN1.bas"
"EITAN2.bas"
"EITAN3.bas"
"EITAN4.bas"
```

そして、ドライブ2のフロッピィディスクに書き込んだデータファイルは

```
EITAN1.dat
```

という、英単語と日本語を書き込んだものでした。

さて、今後あなたがこれらのプログラムを使おうとする場合、いまのままですと、どのプログラムがどんな目的で使われるかを覚えておいて、次のような操作によらなければなりません。

```
run "EITAN1.bas"

run "EITAN2.bas"

run "EITAN3.bas"

run "EITAN4.bas"
```

ファイル名を打ち間違えて

```
File not found
```

というエラーメッセージが返ってきてギクッとしたり、いや、それよりとにかくキーインもめんどうです。もう少し楽をすることはできないでしょうか？

人間は1つのことをマスターすると、そのことをもう少し手をかけないでできないものかとムシのよい考え方をします。しかし、これはおおいにけっこうです。こういった欲がでてきてはじめて前進していくのですから……。

私たちもおおいに思いをめぐらせましょう。

メニュープログラム

そのためには、「英単語練習プログラム」を管理するメニュープログラムを作ってしまうことです。

メニュープログラムとは、画面にいろいろな仕事内容を表示させ、その中からどれか1つを選び、希望する仕事を処理していくものです。

たとえば、メニュー画面を次のように構成します。

```
        エイタンゴ゛ メニュー プ゜ログ゛ラム

     タンゴ゛ ノ カキコミ・・・・・・・・・・1
     タンゴ゛ ノ ヨミダ゛シ・・・・・・・・・2
     タンゴ゛ ノ ツイカ・・・・・・・・・・・3
     オワリ・・・・・・・・・・・・・・・・・4

           input job number ? ■
```

このようにしておきますと、選択した番号に応じて、

```
1 ==> RUN "EITAN1.bas"
```

```
2 ==> RUN "EITAN2.bas"
```

```
3 ==> RUN "EITAN4.bas"
```

というように、それぞれのプログラムが自動的に実行されるわけです。

それではさっそくコーディングしてみましょう。

まず出だしのREM文から。

```
10 ' file name ==> menu01.bas
20 ' 1984.1.25
30 ' Menu program for English words
40 ' save "menu01.bas"
50 '
```

次に初期設定をしておきます。

```
100 CONSOLE 0,25,0,0:WIDTH 40,25
110 DEFINT A-Z ←─────────── 整数宣言
120 CLS
130 '
```

行番号100のWIDTH文で画面のスクロール行数を25にしてありますが、これはメニュー画面の見出しをケイ囲みにするためにこのようにしました。と申しますのは、スクロール行数が20では行間があいてしまってケイ線がつながらないからです。

次に行番号110のDEFINT文ですが、これは変数の型宣言です。 DEFINT

普通このように型の宣言がされていませんと、変数はすべて単精度型とみなされ、処理時間に大きな差がでてきます。数値を扱う場合も整数が主体でしたら、A～Zで始まる変数を整数宣言しておいて、単精度、倍精度扱いの変数にだけ「!」「#」をつけておけば能率的です。あるいは、DEFSNG、DEFDBLなどを使って、あらたに宣言しなおしてもよいでしょう。 DEFSNG DEFDBL

このプログラムではあまり必要ありませんが、処理時間の短縮という意味で、習慣づけておいたほうがよいものの1つです。

次のようなテストを、ダイレクトでいろいろ行ってみてください。

```
defint a-z
Ok
B1=3.14159:B2!=3.14159:B3#=3.141592654
Ok
? B1,B2!,B3#
 3             3.14159       3.141592654
Ok
■
```

メニューはDATA文で

画面に表示するメニューはDATA文にします。 DATA文

```
1000 '
1010 '/// data area ///
1020 '
1030 DATA 4
1040 DATA ﾀﾝｺﾞ ﾉ ｶｷｺﾐ･･････････1
1050 DATA ﾀﾝｺﾞ ﾉ ﾖﾐﾀﾞｼ････････2
1060 DATA ﾀﾝｺﾞ ﾉ ﾂｲｶ･･････････3
1070 DATA ｵﾜﾘ･･････････････････4
```

行番号1030の数値4は、メニューの数を表しています。そして、この値をREAD文で最初に読んで変数に入れておけば非常に便利です。このことはあとで説明します。

行番号1040からはデータです。1行1行にしないで、各データをコンマで区切って、1行に3つでも4つでもつなげて書いてもかまいません。ただ、本づくりの体裁上、このようにしました。

文字列のデータの前後に意味のあるスペースがあるとか、データ中にコンマ、ピリオドなどが含まれる場合は、ダブルクォーテーションでそのデータをくくってください。ここで扱っているデータ中の点は、ピリオドではなくナカグロですから、このままでさしつかえありません。

次に、データを読み込む部分です。

```
140 READ MENU
150 DIM M$(MENU)  ←―― 配列の宣言
160 FOR I=1 TO MENU
170 READ M$(I)  ←―― 配列にメニューを読み込む
180 NEXT I
190 '
```

140行で変数MENUにREAD文でメニューの数を読み込みます。

READ文

次に、それぞれのメニューを配列にとりこみますので、配列の宣言をしなければなりません。行番号150で配列を宣言してから、READ文でメニューを読みます。

もうおわかりいただけたと思いますが、DATA文の先頭にデータ数を入れておくと、このように、配列宣言、FOR～NEXT文のループ回数などが、変数を使って自動的に設定できてしまうのです。

DIM文

もしデータ数を変数に代入しておきませんと、プログラムの初期設定部分でDIM文を入れておかなければなりませんし、FOR文にも数値を入れなければなりません。たとえば次のように……。

```
115 DIM M$(20)
116 '
160 FOR I=1 TO 4
170 READ M$(I)
180 NEXT I
```

〈注〉このプログラムは例ですから入力しないでください。

メニュー数を使ったFOR～NEXT文は他の部分でもでてくると思いますし、メニューを追加するときなどに、いちいちこの数値を修正しておかなければなりません。短いプログラムでしたら、手間はかか

りませんが、長くなるとけっこうめんどうなものです。

また、最初の説明のように、変数を使って配列宣言したほうが、メモリを有効に使えるのです。

これはディスク操作とは関係ありませんが、プログラミングの定石として覚えておきましょう。

メイン部分のプログラム

メニューを画面に表示して、仕事番号を選択させ、希望のプログラムを実行する部分です。プログラムリストは次のとおりです。

```
200 LOCATE 7,4:PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,5:PRINT "| ｴｲﾀﾝｺﾞ ﾒﾆｭｰ ﾌﾟﾛｸﾞﾗﾑ |"
230 LOCATE 7,6:PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270 LOCATE 8,7+I:PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13:INPUT "input job number ";NO
310 LOCATE 8,15:PRINT M$(NO)
320 LOCATE 18,16:INPUT "OK (y/n)";Y$
330 IF Y$="n" OR Y$="N" THEN CLS:GOTO 200
340 ON NO GOTO 360,370,380,390
350 '
360 RUN "EITAN1.bas"
370 RUN "EITAN2.bas"
380 RUN "EITAN4.bas"
390 CLS
400 LOCATE 15,7:PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
410 END
```

見出しの表示（200〜240）

メニューを画面に表示（260〜280）

仕事番号に対応した行番号へ飛ぶ（340）

プログラムをRUN（360〜380）

行番号200から240までは、見出しの表示です。見出しをケイで巻くと体裁がととのいます。

ここではLOCATE文でカーソル位置を指定し、PRINT文で画面に表示しています。STRING＄関数の書式は次のとおりです。

LOCATE文

STRING＄関数

```
string$(21,"─")
string$(21,149)
```

文字または数値（"─" / 149）

出力する個数（21）

この例は２つとも同じ結果を出力します。

文字指定の場合には、ダブルクォーテーションでくくってください。また字数は１字しか有効ではありませんから、３文字入れても最初の１字だけが出力されます。

●「ー」はマイナス記号ではありません。グラフィックキャラクタです。

数値指定の場合には、その値が0〜255の範囲に限られ、これをキャラクタコードとみなして対応する文字を出力します。この例で「ー」のキャラクタコードは149です。

行番号260から280でメニューを画面に表示します。

ここでもMENUという変数をFOR〜NEXT文に使っていることに注意してください。

また、LOCATE文の位置指定にループ用変数Ｉを使っています。これもかなり便利な方法です。

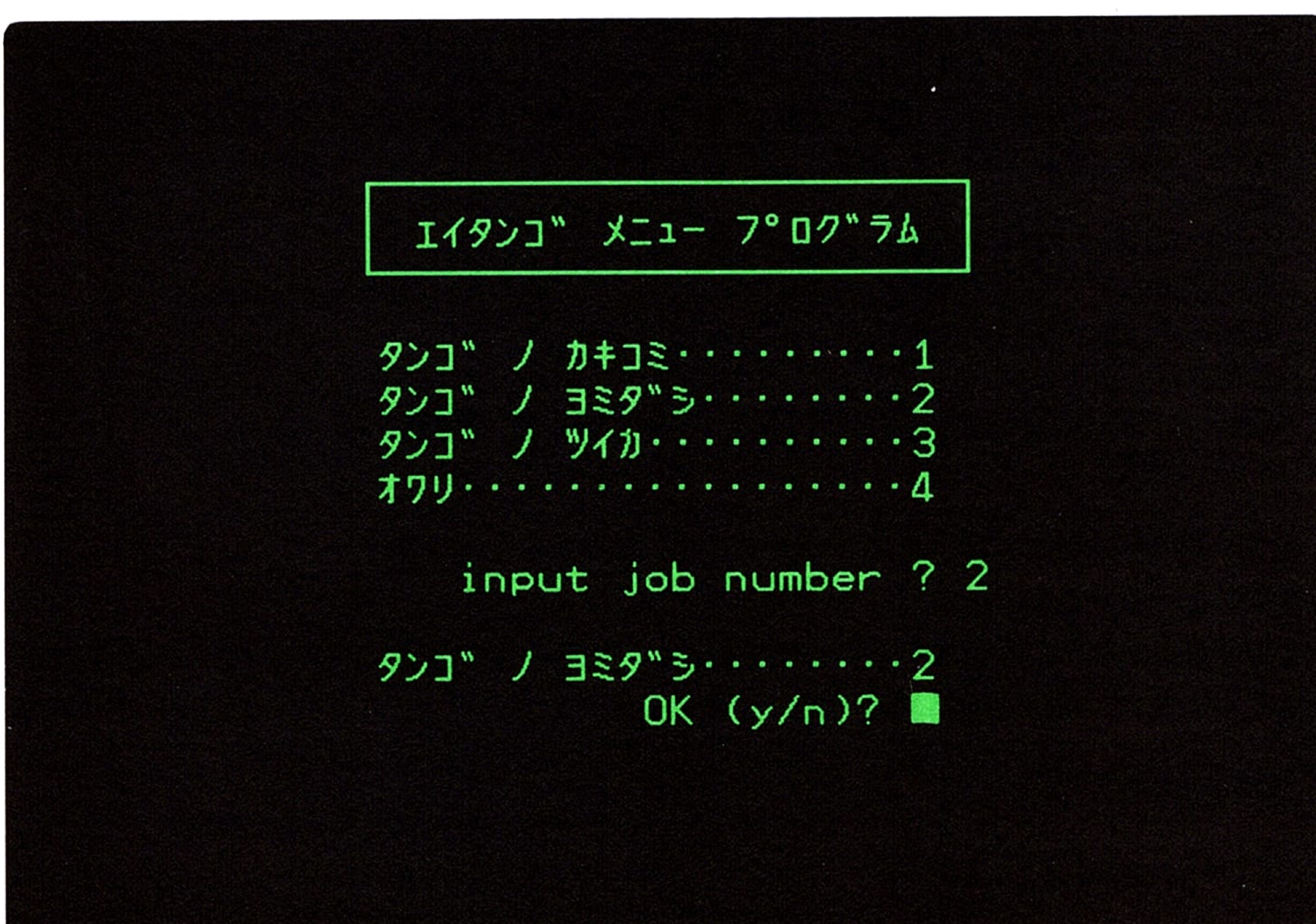
行番号300でメニュー選択をうながし、行番号320では選択したメニューでOKかどうか確認させます。「ノー」であればもう一度200行へ飛ばし、やりなおしです。

ON〜GOTO

OKでしたら、340行のON〜GOTO文で、メニュー番号に対応した処理へ飛びます。

行番号360から380までが、前に説明したプログラム実行のステートメントです。

さあ、完成です。RUNさせてみましょう。



前ページの表示画面は、仕事番号2を選択したので、その確認をうながしているところです。もしよかったら、大文字でも小文字でもよいですからYを入力してみてください。なつかしい「英単語読み出しのプログラム」が走り出します。どうでしたか？　うまくいきましたか？　キーインにミスがなければ必ず動くはずです。

ここで注意を1つ。

メニュー選択の際、1番を選びますと、これはデータ書き込み用プログラムをRUNさせることになりますから、つまり、OUTPUTモードですから、新規にデータファイルをOPENすることになり、いままでのデータがすべて消えてしまいます。せっかく書き込んだものは保存しておきたいですね。

1番を選択するのは新しいファイルを作るときだけです。

ここでもう少し欲を出してプログラミングのテクニックを学んでおきましょう。

引き続き他のプログラムを実行したい

私たちがこの節で組んできたプログラムは、たとえば「単語追加」のメニューを選択しますと、追加処理が終るとプログラムはENDになり、追加したデータを読み出したいときは、もう一度メニュープログラムをロードしてRUNさせなければなりません。

何かよい方法はないでしょうか？

あります、あります。それぞれのプログラムにメニューへ戻るステートメントを折り込んでやればよいのです。

「書き込み」「読み込み」「追加」の3つのプログラムのENDがある行番号は次のとおりでした。

```
EITAN1.bas
        ==>  210 END

EITAN2.bas
        ==>  210 END

EITAN4.bas
        ==>  210 CLOSE:END
```

ですから、これらのEND文を削除して（CLOSE文は残しておく）、このうしろにメニューへ戻るステートメントを書くことにします。プログラムを次に示します。まずNEWコマンドをダイレクトモードで

実行してからキーインしてください。

```
300 '
310 CLS
320 LOCATE 10,6:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 15,8:PRINT "( y/n ) ?"
340 Y$=INKEY$
350 IF Y$="" THEN GOTO 340
360 IF Y$="Y" OR Y$="y" THEN RUN "menu01.bas"
370 END
```

簡単なプログラムですね？

行番号320、330でメッセージを表示しています。

行番330のPRINT文のところはINPUT文を使ってもいいのですが、新しい知識を吸収するという意味で、あえてPRINT文にし、340行でINKEY＄関数を使用しました。

INKEY＄関数

この関数は何かキー入力があるまで、行番号340と350の間をループしています。どのキーでもよいですから、キーが押されると行番号360へ落ちていきます。

さあ、それではこのプログラム名を「mrg」としてアスキー形式でセーブし、「EITAN1.bas」とつなぎ合わせてみましょう。

```
save "mrg",a          ← アスキーセーブします
Ok
load "EITAN1.bas"     ← プログラムをロードします
Ok
210                   ← 行番号210にEND文がありますので、2、1、0
                        とキーインしてリターンキーを押し、この行を削除します
merge "mrg"           ← 「mrg」をマージしてつなげます
Ok
save "EITAN1.bas"     ← 完成したプログラムをセーブします
Ok
█
```

〈注〉「mrg」にEND文がありますので前のプログラムのEND文は削除してください。

まったく同じ方法で「EITAN2.bas」「EITAN4.bas」についてもマージを行ってください(注)。無事終って「mrg」が不要になりましたら、削除しておきましょう。KILL文を使うのでしたね。

```
kill "mrg" ↵
```

LISTコマンドでリストをとってみましょう。

```
10 ' file name ==> EITAN1.bas
20 ' 1984.1.25
30 ' sequential data write
40 ' save "EITAN1.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 CLS
120 OPEN "2:EITAN1.dat" FOR OUTPUT AS #1
130 INPUT "ｴｲｺﾞ (ｵﾜﾘ=99)";A$
140 IF A$="99" THEN GOTO 200
150 INPUT "      ﾆﾎﾝｺﾞ";B$
160 WRITE #1,A$,B$
170 C=C+1
180 IF C=5 THEN C=0:CLS
190 GOTO 130
200 CLOSE #1
300 '
310 CLS
320 LOCATE 10,6:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 15,8:PRINT "( y/n ) ?"
340 Y$=INKEY$
350 IF Y$="" THEN GOTO 340
360 IF Y$="Y" OR Y$="y" THEN RUN "menu01.bas"
370 END
```

行番号300以降がうまくマージされたようです。

これでこの章は終りです。次章では電話帳プログラムを例にとって、もう少しシーケンシャルファイルに慣れるようにしましょう。

第3章

シーケンシャルファイル[II]



3・1 メニュープログラム

前章では「英単語練習プログラム」を例にとってシーケンシャルファイルの扱い方について説明してきました。

この章では、**電話帳プログラム**を例にとって、シーケンシャルファイルについて、もう少し勉強しておきましょう。

電話帳プログラムとは、名前を入力するとその人の電話番号を表示してくれるものです。考え方は英単語の場合とまったく同じですが、みなさんにもう少し慣れていただくためと、検索テクニックがありますのでとりあげることにしました。また、プリンタも使ってみることにしましょう。

ここで扱う仕事は次の5つです。

❶ **電話帳ファイルの作成**――このプログラムは新規のファイル作成のときのみ使用します。

❷ **電話番号の追加**

❸ **電話番号の修正**――すでにファイルに書き込んであるデータを修正します。

❹ **電話番号の検索**

❺ **電話番号リストの印刷**

メニュー画面は次のようにレイアウトしました。

```
 ┌────────────────────┐
 │ ﾃﾞﾝﾜﾁｮｳ ﾌﾟﾛｸﾞﾗﾑ │
 └────────────────────┘

 ﾃﾞｰﾀ ﾉ ｻｸｾｲ･･････････1
 ﾃﾞｰﾀ ﾉ ﾂｲｶ･･･････････2
 ﾃﾞｰﾀ ﾉ ｼｭｳｾｲ･････････3
 ﾃﾞｰﾀ ﾉ ｹﾝｻｸ･･････････4
 ﾃﾞｰﾀ ﾉ ｲﾝｻﾂ･･････････5
 ｵﾜﾘ･･････････････････6

     input job number ? 1

 ﾃﾞｰﾀ ﾉ ｻｸｾｲ･･････････1
           OK (y/n)? ■
```

メニュープログラムは、前章のメニュープログラムの部分的な手直しとDATA文の追加によって作成することにします。**LOADコマンドで「menu01.bas」をロードしてください。**

まず最初の部分は、ファイル名を変更して次のようにします。

```
10 ' file name ==> menu02.bas
20 ' 1984.1.25
30 ' Menu program for phone number
40 ' save "menu02.bas"
50 '
100 CONSOLE 0,25,0,0:WIDTH 40,25
110 DEFINT A-Z
120 CLS
130 '
```
(新しいファイル名 → menu02.bas)

ここはファイル名とREM文の内容を変えるだけでよいでしょう。

次にメニュー項目が2つふえました。メニュー内容も変わるわけですから、**DATA文**を作っておきましょう。

```
1000 '
1010 '/// data area ///
1020 '
1030 DATA 6
1040 DATA デ゛ータ ノ サクセイ・・・・・・・・・・1
1050 DATA デ゛ータ ノ ツイカ・・・・・・・・・・・2
1060 DATA デ゛ータ ノ シュウセイ・・・・・・・・・3
1070 DATA デ゛ータ ノ ケンサク・・・・・・・・・・4
1080 DATA デ゛ータ ノ インサツ・・・・・・・・・・5
1090 DATA オワリ・・・・・・・・・・・・・・・・・・・・6
```
(1030 DATA 6 ← データ数)

データ数増加にともなって、行番号340行以降に**ステートメントの追加・変更**をします。

```
340 ON NO GOTO 360,370,380,381,382,390
350 '
360 RUN "TELNO1.bas"
370 RUN "TELNO2.bas"
380 RUN "TELNO3.bas"
381 RUN "TELNO4.bas"
382 RUN "TELNO5.bas"
390 CLS
400 LOCATE 15,7:PRINT "シコ゛ト ヲ オワリマス"
410 END
```
(381,382 ← 追加; 360〜382 ← 新しいプログラムファイル名)

2つふえたメニューに対するプログラムの実行ステートメントは、行番号380と390の間に入れます。そしてRENUMコマンドで行番号を整理しておきましょう。

```
renum 390,381
```

```
renum 1000,440
```

あとは、見出しの変更と、LOCATE文を修正して表示位置をなおしておきます。

メニューが2つふえましたから、行番号200から280までのLOCATE文の縦座標位置をマイナス2して、2行分上へ表示することにしましょう。カーソル移動キーを使えば簡単に修正できますね。

```
200 LOCATE 7,2:PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,3:PRINT "|   ﾃﾞﾝﾜﾁｮｳ ﾌﾟﾛｸﾞﾗﾑ    |"
230 LOCATE 7,4:PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270 LOCATE 8,5+I:PRINT M$(I)
280 NEXT I
290 '
```

見出しの修正

これでメニュープログラムは完成です。プログラムリストを次に示します。

```
10 ' file name ==> menu02.bas
20 ' 1984.1.25
30 ' Menu program for phone number
40 ' save "menu02.bas"
50 '
100 CONSOLE 0,25,0,0:WIDTH 40,25
110 DEFINT A-Z
120 CLS
130 '
140 READ MENU
150 DIM M$(MENU)
160 FOR I=1 TO MENU
170 READ M$(I)
180 NEXT I
190 '
```

```
200 LOCATE 7,2:PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,3:PRINT "│   ﾃﾞﾝﾜﾁｮｳ ﾌﾟﾛｸﾞﾗﾑ    │"
230 LOCATE 7,4:PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270 LOCATE 8,5+I:PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13:INPUT "input job number ";NO
310 LOCATE 8,15:PRINT M$(NO)
320 LOCATE 18,16:INPUT "OK (y/n)";Y$
330 IF Y$="n" OR Y$="N" THEN CLS:GOTO 200
340 ON NO GOTO 360,370,380,390,400,410
350 '
360 RUN "TELNO1.bas"
370 RUN "TELNO2.bas"
380 RUN "TELNO3.bas"
390 RUN "TELNO4.bas"
400 RUN "TELNO5.bas"
410 CLS
420 LOCATE 15,7:PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
430 END
1000 '
1010 '/// data area ///
1020 '
1030 DATA 6
1040 DATA ﾃﾞｰﾀ ﾉ ｻｸｾｲ··········1
1050 DATA ﾃﾞｰﾀ ﾉ ﾂｲｶ···········2
1060 DATA ﾃﾞｰﾀ ﾉ ｼｭｳｾｲ·········3
1070 DATA ﾃﾞｰﾀ ﾉ ｹﾝｻｸ··········4
1080 DATA ﾃﾞｰﾀ ﾉ ｲﾝｻﾂ··········5
1090 DATA ｵﾜﾘ··················6
```

実行してみたい方は、360行から400行までのステートメントを次のようにREM文にしてRUNさせてみてください。

アポストロフィを入れてREM文にします

```
360 'RUN "TELNO1.bas"
```

正常に動いたらアポストロフィをとってセーブしておきましょう。

```
save "menu02.bas" ⏎
```

3・2　電話帳ファイルの作成

電話帳ファイルの作成プログラムは、データを次々に入力し、それをディスクに書き込めばよいわけですから、67ページのプログラムにちょっと手を加えればできてしまいます。

OUTPUTモードで**OPENするファイル名と、INPUT 文のプロンプト文を変えればよい**でしょう。

次のようなプログラムになります。

```
10 ' file name ==> TELNO1.bas
20 ' 1984.1.25
30 ' sequential data write
40 ' save "TELNO1.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 CLS
120 OPEN "2:TELNO1.dat" FOR OUTPUT AS #1
130 INPUT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ(ｵﾜﾘ=99)";A$
140 IF A$="99" THEN GOTO 200
150 INPUT " input phone number";B$
160 WRITE #1,A$,B$
170 C=C+1
180 IF C=5 THEN C=0:CLS
190 GOTO 130
200 CLOSE #1
300 '
310 CLS
320 LOCATE 10,6:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 15,8:PRINT "( y/n ) ?"
340 Y$=INKEY$
350 IF Y$="" THEN GOTO 340
360 IF Y$="Y" OR Y$="y" THEN RUN "menu02.bas"
370 END
```

ファイル名

データファイル名

プロンプト文を変更

メニュープログラム名

変更したところは、上の画面でコメントを入れてある部分です。あとはすべてそのままです。

まだ実行しなくてけっこうです。データ読み出しの節で合わせて行うことにしましょう。セーブを忘れないように……。

```
save "TELNO1.bas" ↵
```

3·3 データの追加

この部分も、前章の「EITAN4.bas」を手直しして使えそうですね。修正したプログラムリストを示します。

```
10 ' file name ==> TELNO2.bas
20 ' 1984.1.25
30 ' sequential data append
40 ' save "TELNO2.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 CLS
120 GOSUB *DAT.READ
130 DIM AA$(5),BB$(5)
140 OPEN "2:TELNO1.dat" FOR APPEND AS #1
150 FOR I=1 TO 5
160 INPUT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ(ｵﾜﾘ=99)";AA$(I)
165 IF AA$(I)="99" AND I=1 THEN GOTO 210
170 IF AA$(I)="99" AND I<>1 THEN N=I-1
    :GOSUB *DAT.CORRECT:GOTO 210
180 INPUT "   input phone number";BB$(I)
190 NEXT I:N=I-1
200 GOSUB *DAT.CORRECT:GOTO 150
210 CLOSE:END
300 '
310 CLS
320 LOCATE 10,6:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
330 LOCATE 15,8:PRINT "( y/n ) ?"
340 Y$=INKEY$
350 IF Y$="" THEN GOTO 340
360 IF Y$="Y" OR Y$="y" THEN RUN "menu02.bas"
370 END
1000 '
1010 *DAT.READ
1020 '
1030 LOCATE 10,5:PRINT "ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ"
1040 OPEN "2:TELNO1.dat" FOR INPUT AS #1
1050 IF EOF(1) THEN GOTO 1090
1060 INPUT #1,A$,B$
1070 C=C+1
1080 GOTO 1050
```

ファイル名（10行・40行の「TELNO2.bas」）

データファイル名（140行の「2:TELNO1.dat」）

プロンプト文を修正（160行・180行）

メニュープログラム（360行の「menu02.bas」）

わずらわしいのでこのようにしました（1010〜1020行）

データファイル名（1040行の「2:TELNO1.dat」）

```
1090 CLOSE #1
1100 PRINT
1110 LOCATE 7,7:PRINT "ﾏﾂﾋﾞ ﾉ ﾃﾞｰﾀ==> ";A$
1120 LOCATE 7,8:PRINT "ﾄｳﾛｸ ｿｳｽｳ  ==>";C;" ﾒｲ"
1130 PRINT
1140 LOCATE 12,10:INPUT "OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ";R$
1150 CLS
1160 RETURN
2000 '
2010 *DAT.CORRECT
2020 '
2030 CLS
2040 LOCATE 5,1:PRINT "ﾆｭｳﾘｮｸ ﾃﾞｰﾀ ｲﾁﾗﾝ"
2050 FOR J=1 TO N
2060 LOCATE 2,2+2*J:PRINT AA$(J)
2070 LOCATE 10,3+2*J:PRINT BB$(J)
2080 NEXT J
2090 LOCATE 7,2
2100 INPUT "ｼｭｳｾｲ ｼﾏｽｶ  ( y/n )";ANS$
2110 IF ANS$="n" OR ANS$="N" THEN GOTO 2180
2120 LOCATE 5,1:PRINT "ｼｭｳｾｲ ｶﾞ ﾅｹﾚﾊﾞ ｶｰｿﾙ ｦ"
2130 LOCATE 5,2:PRINT "ﾃﾞｰﾀﾉ ﾐｷﾞﾄﾞﾅﾘﾍｲﾄﾞｳｼﾃ RET"
2140 FOR J=1 TO N
2150 LOCATE 0,2+2*J:INPUT "";AA$(J)
2160 LOCATE 8,3+2*J:INPUT "";BB$(J)
2170 NEXT J
2180 FOR J=1 TO N
2190 WRITE #1,AA$(J),BB$(J)
2200 NEXT J
2210 CLS
2220 RETURN
```

メッセージ変更

ここも変更しました

変更・修正したところにはコメントを入れてあります。

これで、「データの追加」部分もできてしまいました。意外と簡単ですね。このように、自分に関連した仕事のプログラムを作る場合、できるだけ同じようなプログラムを修正・追加して流用すると、バグも少なくてすみますし、手間もはぶけます。

修正が終ったらセーブしておきましょう。

```
save "TELNO2.bas"
```

3・4　登録・追加プログラムの実行

この節では、すでに作成ずみの電話番号書き込み用・追加用のプログラムファイルを実行してテストをしてみます。また、この結果作られたデータファイル「TELNO1.dat」は次節以降で使うことにいたしましょう。

まず、ドライブ1のディスクにはメニュープログラムが保存されていますから、これをロードして実行するところから始めます。

```
run "menu02.bas" ⏎
```

70ページのようなディスプレイ画面になりましたか？

いま私たちはデータの書き込みをしようとしているわけですから、仕事番号は「1」をキーインします。そして ⏎ です。

すると、入力に対しての確認を求めてきますから、もしよければyをキーインし ⏎ キーを押します。

すぐにデータ書き込みのプログラム「TELNO1.bas」がロードされ、続いて実行されます。画面がクリアされてデータ入力をうながすメッセージが表示されますから、テストデータを入力してみましょう。

次の4名について入力してみましょう。

```
トクガ゛ワ  イエヤス  ==>  000-000-0000
トヨトミ  ヒデ゛ヨシ  ==>  111-111-1111
オダ゛  ノブ゛ナガ゛  ==>  222-222-2222
アケチ  ミツヒデ゛    ==>  333-333-3333
```

4人の氏名と電話番号を打ち込み、5人目のメッセージが表示されたら「99」を入力し ⏎ キーを押します。

```
ナマエ ヲ ド゛ウゾ゛(オワリ=99)? トクガ゛ワ イエヤス
 input phone number? 000-000-0000
ナマエ ヲ ド゛ウゾ゛(オワリ=99)? トヨトミ ヒデ゛ヨシ
 input phone number? 111-111-1111
ナマエ ヲ ド゛ウゾ゛(オワリ=99)? オダ゛ ノブ゛ナガ゛
 input phone number? 222-222-2222
ナマエ ヲ ド゛ウゾ゛(オワリ=99)? アケチ ミツヒデ゛
 input phone number? 333-333-3333
ナマエ ヲ ド゛ウゾ゛(オワリ=99)? ■
```

← 「99」を入力しリターンキーを押す

次に画面には「ほかの仕事をするかどうか」をたずねるメッセージが表示されますから、「y⏎」と入力します。するとすぐにメニュー画面に戻ります。

今度は追加処理メニュー２を選択してみましょう。１を選択したときと同じ手順で次の画面表示に至ります。このようになっていれば、書き込みは順調にいっているはずです。

```
      ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ

 ﾏﾂﾋﾞ ﾉ ﾃﾞｰﾀ==> ｱｹﾁ ﾐﾂﾋﾃﾞ
 ﾄｳﾛｸ ｿｳｽｳ   ==> 4   ﾒｲ

        OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ? ■
```

どうですか？　間違いなくできていましたか？

それでは ⏎ キーを押して次へ進みましょう。

以下の操作は２・７節で勉強ずみですので、それほど苦もなくできると思います。次の６人の電話番号を追加しておきましょう。

サンプルデータの電話番号はきれいにそろっていますが、これは見やすくするためです。何桁でもかまいません。

```
ｱﾗｲ ﾊｸｾｷ      ==> 444-444-4444
ｶｲﾊﾞﾗ ｴｯｹﾝ    ==> 555-555-5555
ｶｲﾎ ｾｲﾘｮｳ     ==> 666-666-6666
ﾖｼﾀﾞ ｹﾝｺｳ     ==> 777-777-7777
ｻﾄｳ ｲｯｻｲ      ==> 888-888-8888
ﾖｼﾀﾞ ｼｮｳｲﾝ    ==> 999-999-9999
```

(注)ストップキーでプログラムを停止させたときには、ダイレクトモードでCLOSEとキーインし、リターンキーを押しておきましょう。この場合はだいじょうぶですが、ファイルがOPENされたままのときもあります。

６人のデータ入力が終りましたら「99」を入力し、「ホカノシゴトヲシマスカ」というメッセージに対しては「n」をキーインし、本節でのテストを終ります。データ追加の確認をしたい方は、この状態で f･5 キーを押し、次の画面で STOP キー(注)を押せば確かめられます。

```
      ﾃﾞｰﾀ ﾉ ﾂｲｶ ｦ ｼﾏｽ

 ﾏﾂﾋﾞ ﾉ ﾃﾞｰﾀ==> ﾖｼﾀﾞ ｼｮｳｲﾝ
 ﾄｳﾛｸ ｿｳｽｳ   ==> 10  ﾒｲ  ←確かに登録されていました

        OK ﾅﾗ RETｷｰ ｦ ﾄﾞｳｿﾞ? ←ここでリターンキーを押さずにストップキーを押す
Break in 1140
Ok
■
```

3·5 データの検索

ここまで進んできたあなたは、もうだいぶシーケンシャルファイルの扱いに慣れたことと思います。私たちが日常かかわっている実務上のファイルと違ってディスクのデータファイルは、いくら書き込まれたといっても、目に見えないものですから実際に読み出して確認してみるまでは不安が残ります。初めのうちはとくに、書き込みがうまくいったのかどうか、追加は確実にできているのかどうか、ほんとうに心配なものです。

しかし、たまには失敗もあるかもしれませんが、実践を積み重ねていくうちにだんだん自信がもてるようになってくると思います。

さて、この節では、初めてのテーマ「データの検索」をマスターすることにしましょう。

読んで字のごとく、データを探すこと、自分のほしいデータをコンピュータに探させ画面に表示させようというわけです。

シーケンシャルファイルとランダムファイルの違いのところで述べましたが、シーケンシャルファイルでは、ある1つのデータだけを直接に読み出すことができません。最初のデータからたんねんに読んでいかなければならないのです。ずいぶん融通がきかないと思いますが、コンピュータがやってくれるのですから、私たちはだまって従うことにいたしましょう。

では、いつものとおり出だしの部分から始めましょう。

```
10 ' file name ==> TELNO4.bas
20 ' 1984.1.25
30 ' sequential data search
40 ' save "TELNO4.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
120 CLS
130 GOSUB *DAT.READ1   ←データ読み出しルーチンへ
```

ファイル名は、これまでの順序でいくと「TELNO3.bas」ですが、データ検索はメニュー画面で仕事番号4にしてしまいましたので、このようにしました。

行番号130のGOSUB文でジャンプさせ、その飛び先でデータファイルからデータを読み出し、検索の準備をします。

ここでのデータ検索の方法ですが、ファイルから読み出したデータを配列変数にしまっておいて、画面のメッセージに従って探したい人の名前がキーボードから入力されたら、それと配列にしまわれている文字列とを比較・判別することによって該当するデータを表示するようにします。

配列の有効な使い方

そこでまず配列を宣言しなければならないわけですが、これもあらかじめ

```
DIM AA$(200),BB$(200)
```

というように、一定の大きさで確保しておいてもよいのですが、もし登録されているデータ数が50だとすると、残りのメモリは何の役にもたっていないわけですから、ずいぶんムダな使い方をしていることになります。

これを防ぐ方法として、まず最初に、配列変数ではなく普通の文字変数（ダミー変数）でデータを読みだしてデータ数をカウントし、その数を配列の大きさに設定することです。1000行からのサブルーチンでこの処理をすることにしましょう。

```
1000 '
1010 *DAT.READ1
1020 '
1030 OPEN "2:TELNO1.dat" FOR INPUT AS #1
1040 WHILE NOT(EOF(1))
1050 INPUT #1,A$,B$
1060 C=C+1 ←――――――――――――データ数をカウントする
1070 WEND
1080 CLOSE #1
1090 RETURN
```

変数Cをカウンタにして、A＄、B＄という文字変数を使って、氏名、電話番号の2つひと組のデータを読み出すたびに1ずつカウントしています。

WHILE〜WEND

データ末尾のチェックはWHILE〜WEND文を使っています。もしEOF、つまりファイルの中に読み出すデータがなくなれば、このループから脱出し、RETURN文でGOSUB文のすぐうしろに戻りますから、そこで配列の宣言をします。

配列の宣言

```
140 DIM AA$(C),BB$(C)
```

配列の大きさが決まりましたので、もう一度ファイルをOPENしてデータを読み出し配列の中にしまわなければなりません。二度手間にはなりますが、このほうがメモリの有効利用という点ではずいぶんプ

ラスになるのです。

配列へのデータ読み込みもサブルーチンにして、行番号を2000からとします。まず150行にGOSUB文を入れ、次にサブルーチンをコーディングします。

```
150 GOSUB *DAT.READ2
```

```
2000 '
2010 *DAT.READ2
2020 '
2030 OPEN "2:TELNO1.dat" FOR INPUT AS #1
2040 FOR I=1 TO C
2050   INPUT #1,AA$(I),BB$(I)
2060 NEXT I
2070 CLOSE #1
2080 RETURN
```

この場合EOF文を入れる必要はありません。というのはすでに変数Cにデータ数が入っていますから、この数を超えて読み出そうとすることがないからです。

これで一応下準備ができました。次はメインルーチンです。

まず画面中央に仕事の目的を表示するようにしましょう。

```
160 '
170 LOCATE 10,7:PRINT "デ"ータ ヲ サカ"シマス"
180 FOR I=1 TO 5000:NEXT I
```

次は電話番号を知りたい人の名前を入力する部分です。(カナ) キーを押してカタカナで入力しなければなりませんし、カタカナをキーボードから探すのは慣れるまでけっこう手間どるものです。ここではちょっと楽をして姓だけをキーインし、同姓の場合の処理は別に考えることにします。姓を入力して ⏎ キーを押したら、データ探しのサブルーチンへ飛ぶようにします。

```
190 '
200 CLS
210 LOCATE 8,5:PRINT "ト"ナタ ノ TEL No.ヲ サカ"シマスカ"
220 LOCATE 8,6:PRINT "    input family name"
230 LOCATE 8,7:INPUT "==>";NM$
240 L=LEN(NM$) ←――――― 文字列の長さを求める
250 '
260 GOSUB *DAT.SEARCH ←――――― データサーチサブルーチンへ
```

LEN関数

行番号240にLEN関数があります。これは文字列（ストリング）の長さを求める関数で書式は次のとおりです。

```
L=LEN(A$)
```

A＄という文字列の長さがLに代入されます。文字列の前後にスペースが入っているときは、そのスペースも長さに加えられることに注意してください。

ここでLEN関数を使いましたのは、配列AA$(Ｉ)に読み込まれている名前はフルネームで入っているため、このうち姓の部分だけを取り出して比較するためです。

つまり、AA$(Ｉ)の左からＬによって与えられる長さだけ文字列を取り出せば、それが姓にほかならないわけです。

データ検索のサブルーチンは、これを利用して次のようにプログラミングしました。

```
3000 '
3010 *DAT.SEARCH
3020 '
3030 FOR I=1 TO C
3040 IF NM$=LEFT$(AA$(I),L)
     THEN N=N+1:AC$(N)=AA$(I):BC$(N)=BB$(I)
3050 NEXT I
3060 RETURN
```

配列AA$(Ｉ)の中をデータ数Ｃによって与えられる数だけ探せばよいわけですから、FOR～NEXTループにします。

LEFT$

そして、あなたが入力した名前が登録されているデータの中にあるかどうかの判断は、LEFT$文を使った比較によって行っています。

RIGHT$
MID$

LEFT$文と同じように、ある一定の長さの文字列を取り出す関数にRIGHT$関数、MID$関数があります。

```
A$=LEFT$(B$,L)     ←  B$の左からL個の文字列をA$に代入

A$=RIGHT$(B$,L)    ←  B$の右からL個の文字列をA$に代入

A$=MID$(B$,L,M)    ←  B$の左からL番目より数えてM個の
                      文字列をA$に代入
```

機能については上のコメントで説明していますので省きますが、けっこう便利な関数です。

ダイレクトモードでキーインし、この関数の機能を実際に確かめてみましょう。

```
A$="0123456789"
Ok
B$=LEFT$(A$,5):C$=RIGHT$(A$,5):? B$,C$
01234         56789
Ok
D$=MID$(A$,4,3):? D$
345
Ok
```

上の画面の文字変数はつなげることもできますから、工夫次第でかなりユニークな使い方ができるのではないかと思います。あなたも、プログラミングの中で役立ててみてはいかがでしょうか？

さて、3040行でLEFT$関数で登録されている氏名の左からL文字をとり出し、それをNM$と比較し、2つの文字列がもし同じであれば、THEN文以降のステートメントへ進みます。

電話番号を調べたい人の名前がみつかったわけですから、カウンタNをプラス1しておきます。変数Nは初めは0ですから、1つ見つかれば0＋1＝1、つまり1がNに代入されます。

次に、探しあてた人の氏名と電話番号を新しい配列AC$とBC$に代入しています。この配列は、同姓の人は複数人いる可能性もあるわけですから、同姓の人をすべて退避させておくためのものです。

配列を使った登録方法

たとえば、佐藤さんという人が3人いたとしますと、ファイルに登録してある順に、AC$(1)、AC$(2)、AC$(3)の中に格納されていくわけです。そして、これを再び画面に表示して選択させればよいわけです。

初めの入力のときにフルネームにしておけば、このようなことはあまり起こらないでしょうが、入力が姓のみである上にカタカナの入力ですから、読み方が同じの人はけっこういるかもしれません。

とにかく、プログラミングするときは、いろいろなケースを想定しておくことが大切です。

このサブルーチンからRETURN文によって戻ると、今度は、見つからなかった場合、1名見つかった場合、複数人の場合の処理をすることになります。

この処理過程もやはりサブルーチンにすることにします。

メインルーチンはできるだけ簡素にしておいて、うしろのほうにいろいろなサブルーチンをつけておくことが望ましいと思います。

では、いま説明した3つの場合を処理する部分を次に示します。

```
270 IF N=0 THEN GOSUB *N.FOUND:GOTO 200
280 IF N=1 THEN GOSUB *FOUND1
          :GOSUB *CALL.INKEY:GOTO 310
290 GOSUB *FOUND2
300 '
310 CLS
320 LOCATE 12,5:PRINT "ｹﾝｻｸ ｦ ﾂﾂﾞｹﾏｽｶ"
330 LOCATE 12,6:INPUT "  ( y/n )";ANS$
340 IF ANS$="Y" OR ANS$="y" THEN N=0:GOTO 200
350 LOCATE 12,8:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 12,9:INPUT "  ( y/n )";ANS$
370 IF ANS$="y" THEN RUN "menu02.bas"
380 END
```

見つからない（270行）
1名見つかった（280行）
複数人見つかった（290行）

4000行からのサブルーチンは目的の名前が登録されてなかった場合の処理です。サブルーチンから戻ると、200行へ飛ばしてもう一度入力させます。

5000行からは、1名見つかった場合、6000行からは同姓の人が複数人いた場合の処理です。

キー入力待ちのサブルーチン

●長いプログラムですと何回もキー入力待ちのステートメントが必要になりますので、サブルーチン化しておくと便利です。

7000行からは、キー入力待ちのサブルーチンとしました。短いものですが、複雑なプログラムではかなり使用する機会がありますので、作っておくと便利です。この程度のプログラムではそれほど必要ないかもしれませんが、あとあとのために習慣づけておくとよいでしょう。

310行からは、その後の作業をどうするか、その意志を求めるものです。メニュープログラムへ戻れるようにしておきました。

ここまででメインルーチンは終りです。このようにメイン部分は短くしておくとよいと思います。

次に各サブルーチンをコーディングしていきましょう。

該当者なしの場合

まず、ファイルに登録されていなかったとき。

```
4000 '
4010 *N.FOUND
4020 '
4030 CLS
4040 BEEP 1:FOR I=1 TO 2000:NEXT I:BEEP 0
4050 LOCATE 8,5:PRINT "ｿﾉ ｶﾀ ﾊ ﾄｳﾛｸ ｼﾃｱﾘﾏｾﾝ"
4060 LOCATE 8,7
4070 PRINT "  Start from the beginning!"
4080 FOR I=1 TO 5000:NEXT I
4090 RETURN
```

ブザー音を出して注意します（4040行）
メッセージがすぐ消えないようにするタイマー（4080行）

画面をクリアし、ブザー音とともにメッセージを表示し、注意をうながしています。

この場合は、飛び先へ戻ったら、もう一度入力をしなおします。

該当者１名の場合

次は、１名該当者がいた場合です。

```
5000 '
5010 *FOUND1
5020 '
5030 CLS
5040 LOCATE 10,7:PRINT "TEL No.・・・";BC$(N)
5050 LOCATE 8,9
5060 PRINT "Hit RET key if you are OK!"
5070 RETURN
```

これも、問題ないと思います。配列BC$(N)の添字Nは、１名しかいないのですから１でもかまいません。

該当者が複数人いた場合

次が３番目のケース、複数人いた場合です。

```
6000 '
6010 *FOUND2
6020 '
6030 CLS
6040 LOCATE 8,3
6050 PRINT USING "ﾄﾞｳｾｲ ﾉ ﾋﾄ ｶﾞ ##ﾒｲ ｲﾏｽ";N
6060 FOR I=1 TO N
6070 LOCATE 8,4+I
6080 PRINT USING "## &              &";I,AC$(I)
6090 NEXT I
6100 LOCATE 10,5+I:INPUT "input number";M
6110 CLS
6120 LOCATE 8,5
6130 PRINT AC$(M);" ｻﾝ ﾉ TEL No. ﾊ"
6140 LOCATE 12,6:PRINT BC$(M);"  ﾃﾞｽ"
6150 LOCATE 8,8
6160 PRINT "Hit RET key if you are OK!"
6170 GOSUB *CALL.INKEY
6180 RETURN
```

スペース14個（6080行）
フルネームを表示（6080行）
番号を入力（6100行）
電話番号を表示（6130〜6140行）

同姓の人が２名以上いた場合には、すべて画面にフルネームで表示し、その中から目的の名前を番号で入力させます。番号をキーインして ⏎ キーを押すと、すぐに電話番号が表示されます。

6160、6170行はキー入力待ちの処理です。⏎キーが押されるまで表示画面がそのまま維持されます。

PRINT USING

6050、6080行にあるPRINT USING文について簡単に説明しておきます。ごく一般的に使われているのは次の3つです。

```
❶ PRINT USING "#####";A        ← 数値を出力する桁数を指定
❷ PRINT USING "#####,";A       ← 3桁ごとにコンマでくぎる
❸ PRINT USING "&           &";A$ ← 文字を左づめで出力する
```

❶は数値を出力する桁数を指定するものです。数値が指定されている桁数（例では5桁）より小さいときは、右づめで出力されます。したがって、縦に並んで出力される数値を右ゾロエにするのによく使われます。

❷は3桁ごとにコンマが入るため、大きな数値を扱うときに見やすくなります。

❸は左の＆(アンド)マークから右の＆マークまでの字数で（例では11文字）文字列を出力します。この字数より長ければ余分なものは無視されますし、短かければ左づめで出力されます。

ダイレクトモードでいくつかためしてみてください。

```
a=1234:b!=123456:a$="PC-8801"
Ok
print using "#####";a
 1234
Ok
print using "#######,";b!
 123,456
Ok
print using "    &           &";a$
    PC-8801
Ok
■
```

（注：`"`と`&`の間は4スペースあいています）
（注：スペースが4つ入り、5文字目から左づめで出力される）

PRINT USING文にはまだいろいろな使い方がありますが、普通はこの程度で間に合ってしまいます。

ここで紹介しなかったものはマニュアルに載っていますから、使う機会が生じましたら、参照してください。

次は最後のサブルーチンです。

```
7000 '
7010 *CALL.INKEY
7020 '
7030 Y$=INKEY$
7040 IF Y$="" THEN GOTO 7030 ELSE RETURN
```

これで完成しました。プログラムをセーブしてから実行してみましょう。

RUNさせますと次のような画面になります。

```
ﾄﾞﾅﾀ ﾉ TEL No.ｦ ｻｶﾞｼﾏｽｶ
   input family name
==>? ■
```

姓を入力してみてください。見つからなければ、ピーッとブザー音がして次のメッセージが表示されます。

```
ｿﾉ ｶﾀ ﾊ ﾄｳﾛｸ ｼﾃｱﾘﾏｾﾝ

  Start from the beginning!
```

入力画面になりましたら「ﾖｼﾀﾞ」とキーインしてください。

```
ﾄﾞｳｾｲ ﾉ ﾋﾄ ｶﾞ  2ﾒｲ ｲﾏｽ

 1 ﾖｼﾀﾞ ｹﾝｺｳ
 2 ﾖｼﾀﾞ ｼｮｳｲﾝ

  input number? ■
```

← 番号を入力する

異常なさそうですね。

最後は次の画面です。

```
ｹﾝｻｸ ｦ ﾂﾂﾞｹﾏｽｶ
  ( y/n )? n

ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ
  ( y/n )? ■
```

← 終りならnを入力

← yを入力すればメニュー画面へ

```
10 ' file name ==> TELNO4.bas
20 ' 1984.1.25
30 ' sequential data search
40 ' save "TELNO4.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
120 CLS
130 GOSUB *DAT.READ1
140 DIM AA$(C),BB$(C)
150 GOSUB *DAT.READ2
160 '
170 LOCATE 10,7:PRINT "ﾃﾞｰﾀ ｦ ｻｶﾞｼﾏｽ"
180 FOR I=1 TO 5000:NEXT I
190 '
200 CLS
210 LOCATE 8,5:PRINT "ﾄﾞﾅﾀ ﾉ TEL No.ｦ ｻｶﾞｼﾏｽｶ"
220 LOCATE 8,6:PRINT "    input family name"
230 LOCATE 8,7:INPUT "==>";NM$
240 L=LEN(NM$)
250 '
260 GOSUB *DAT.SEARCH
270 IF N=0 THEN GOSUB *N.FOUND:GOTO 200
280 IF N=1 THEN GOSUB *FOUND1
         :GOSUB *CALL.INKEY:GOTO 310
290 GOSUB *FOUND2
300 '
310 CLS
320 LOCATE 12,5:PRINT "ｹﾝｻｸ ｦ ﾂﾂﾞｹﾏｽｶ"
330 LOCATE 12,6:INPUT "   ( y/n )";ANS$
340 IF ANS$="Y" OR ANS$="y" THEN N=0:GOTO 200
350 LOCATE 12,8:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 12,9:INPUT "   ( y/n )";ANS$
370 IF ANS$="y" THEN RUN "menu02.bas"
380 END
1000 '
1010 *DAT.READ1
1020 '
1030 OPEN "2:TELNO1.dat" FOR INPUT AS #1
1040 WHILE NOT(EOF(1))
1050 INPUT #1,A$,B$
1060 C=C+1
1070 WEND
1080 CLOSE #1
1090 RETURN
2000 '
2010 *DAT.READ2
2020 '
2030 OPEN "2:TELNO1.dat" FOR INPUT AS #1
2040 FOR I=1 TO C
2050   INPUT #1,AA$(I),BB$(I)
2060 NEXT I
```

```
2070 CLOSE #1
2080 RETURN
3000 '
3010 *DAT.SEARCH
3020 '
3030 FOR I=1 TO C
3040 IF NM$=LEFT$(AA$(I),L)
     THEN N=N+1:AC$(N)=AA$(I):BC$(N)=BB$(I)
3050 NEXT I
3060 RETURN
4000 '
4010 *N.FOUND
4020 '
4030 CLS
4040 BEEP 1:FOR I=1 TO 2000:NEXT I:BEEP 0
4050 LOCATE 8,5:PRINT "ｿﾉ ｶﾀ ﾊ ﾄｳﾛｸ ｼﾃｱﾘﾏｾﾝ"
4060 LOCATE 8,7
4070 PRINT "  Start from the beginning!"
4080 FOR I=1 TO 5000:NEXT I
4090 RETURN
5000 '
5010 *FOUND1
5020 '
5030 CLS
5040 LOCATE 10,7:PRINT "TEL No.･･･";BC$(N)
5050 LOCATE 8,9
5060 PRINT "Hit RET key if you are OK!"
5070 RETURN
6000 '
6010 *FOUND2
6020 '
6030 CLS
6040 LOCATE 8,3
6050 PRINT USING "ﾄﾞｳｾｲ ﾉ ﾋﾄ ｶﾞ ##ﾒｲ ｲﾏｽ";N
6060 FOR I=1 TO N
6070 LOCATE 8,4+I
6080 PRINT USING "## &              &";I,AC$(I)
6090 NEXT I
6100 LOCATE 10,5+I:INPUT "input number";M
6110 CLS
6120 LOCATE 8,5
6130 PRINT AC$(M);" ｻﾝ ﾉ TEL No. ﾊ"
6140 LOCATE 12,6:PRINT BC$(M);"  ﾃﾞｽ"
6150 LOCATE 8,8
6160 PRINT "Hit RET key if you are OK!"
6170 GOSUB *CALL.INKEY
6180 RETURN
7000 '
7010 *CALL.INKEY
7020 '
7030 Y$=INKEY$
7040 IF Y$="" THEN GOTO 7030 ELSE RETURN
```

3・6 データの修正

第２章の「英単語練習プログラム」では、シーケンシャルファイルにすでに書き込んだデータの修正は扱いませんでした。

しかし電話番号のファイルとなると、たまには番号変更もあるでしょうし、また姓が変わったりする人もいます。ですから、ある人の名前を入力したら、そのデータを検索して氏名・電話番号ともに、自由に修正することができたら大変便利だと思いませんか？

この章では、この辺を学んでいこうというわけです。

データ修正の手順

シーケンシャルファイルにおけるデータ修正は、まず初めに、現在すでにフロッピィディスクに書き込まれているデータを、一度すべて読み出し、それを配列にしまっておきます。

そして、ディスプレイ上のメッセージに従って、キーボードから変更したい人の名前を入力します。次に、この入力データをもとに配列の中から探し出し、これをディスプレイに表示させ、正しい氏名、電話番号を入力させ、配列の中味を入れかえてしまいます。

この操作を修正の必要のあるデータ数だけくりかえし、最後に修正された配列の中味を、データファイルを新規にOPENして書き込みます。これで修正されたマスターファイルの出来上がりです。

もう、お気づきのことと思いますが、データ読み出しのところ、またプログラムの全体的な流れなど、前節のデータ検索とだいぶ共通しています。そんなわけで、データサーチのプログラムに手を加えたり、あるいは、追加サブルーチンなどを作ったりして完成させていくことにいたしましょう。

まず前節のプログラムをロードしてください。そして、できるだけスクリーンエディタでなおしていきましょう。

```
load "TELNO4.bas"
Ok
■
```

まず初めは、プログラムの見出し部分です。

前節のプログラム名と順序が前後しますが、ファイル名「TELNO3.bas」としました。

```
10 ' file name ==> TELNO3.bas
20 ' 1984.1.25
30 ' sequential data correct
40 ' save "TELNO3.bas"
50 '
```

行番100から160までの初期設定部分は「TELNO4.bas」と同じです。

次はメインルーチン部分です。　　メインルーチン

```
170 LOCATE 10,7:PRINT "デ゛ータ ノ シュウセイ ヲ シマス"
180 FOR I=1 TO 5000:NEXT I
190 '
200 CLS
210 LOCATE 8,5:PRINT "ド゛ナタ ノ デ゛ータ ヲ シュウセイ シマスカ"
220 LOCATE 8,6:PRINT "    input family name"
230 LOCATE 8,7:INPUT "==>";NM$
240 L=LEN(NM$)
250 '
260 GOSUB *DAT.SEARCH
270 IF N=0 THEN GOSUB *N.FOUND:GOTO 200
280 IF N=1 THEN GOSUB *DAT.CORRECT:GOTO 310
290 GOSUB *DAT.SELECT:GOSUB *DAT.CORRECT    ← 追加
300 '
310 CLS
320 LOCATE 12,5:PRINT "シュウセイ ヲ ツヅ゛ケマスカ"
330 LOCATE 12,6:INPUT "   ( y/n )";ANS$
340 IF ANS$="Y" OR ANS$="y" THEN N=0:GOTO 200
345 GOSUB *DAT.WRITE    ← 追加ステートメント
350 LOCATE 12,8:PRINT "ホカ ノ シゴ゛ト ヲ シマスカ"
360 LOCATE 12,9:INPUT "   ( y/n )";ANS$
370 IF ANS$="y" THEN RUN "menu02.bas"
380 END
```

変更・追加点は緑色の部分です。

行番号280では、7000行からのキー入力待ちルーチンは必要なくなりましたので削除しました。7000行からの削除は、まだ行わないでけっこうです。

5000行からを修正サブルーチンとし、該当データが1名の場合も複数名の場合も、このルーチンを使うことにしました。280行、290行でこのサブルーチンへ飛ばしています。

320、330行で修正を続けるかどうかの確認をし、もし終りにするの

であれば、データ書き込みのサブルーチンへ飛ばします。行番号を345として、GOSUB文を追加しました。

行番号1000および2000からのデータ読み込み用サブルーチンは、まったく同じです。

データ検索サブルーチン　次は3000行からのデータ検索のサブルーチンです。

```
3000 '
3010 *DAT.SEARCH
3020 '
3030 FOR I=1 TO C
3040 IF NM$=LEFT$(AA$(I),L) THEN N=N+1
      :X(N)=I:AC$(N)=AA$(I):BC$(N)=BB$(I)
3050 NEXT I   ↑
3060 RETURN   └──────────────────挿入
```

この部分での変更点は1か所、3040行に代入文を1つ挿入しました。

インデックス　X(N)という配列は一種のインデックスで、AC$(1)、AC$(2)、……の中味が、配列AA$の何番目に対応しているかを示すものです。修正はAC$、BC$という配列を使って行いますので、修正後、その修正データをもとの配列AA$、BB$に戻してやらなければなりません。そのため、戻す先が何番目かを記憶しておき、代入の橋わたしをするのが配列Xです。

検索サブルーチンで修正すべきデータが見つかったら、いよいよ修

データ修正サブルーチン　正作業をしなければなりません。これは5000行からとします。

```
5000 '
5010 *DAT.CORRECT
5020 '
5030 CLS
5040 LOCATE 5,2:PRINT AC$(N);" サン ノ シュウセイ"
5050 LOCATE 5,3
5060 INPUT "   マチカ゛イナイテ゛スカ ( y/n )";ANS$
5070 IF ANS$="N" OR ANS$="n" THEN RETURN
5080 LOCATE 5,4
5090 PRINT "タタ゛シイ テ゛ータ ヲ ト゛ウソ゛(シュウセイ ナシ=0)"
5100 LOCATE 5,5:PRINT "シメイ= ";AC$(N)
5110 LOCATE 5,6:INPUT "    ==> ";NM$
5120 IF NM$<>"0" THEN AA$(X(N))=NM$ ←
5130 LOCATE 5,7:PRINT "TEL No.= ";BC$(N)
5140 LOCATE 5,8:INPUT "    ==>";TEL$
5150 IF TEL$<>"0" THEN BB$(X(N))=TEL$ ←
5160 RETURN      配列AA$、BB$の中味を修正値と入れ換え
```

このサブルーチンは全面的に変わります。このリストどおりにキーインしてください。前のプログラムの5000行からは削除しなくてもかまいません。あとから入力した行番号のステートメントが優先されますから、自動的に切り換わります。

5060、5070行では、画面に表示された人の修正でよいかどうか確認しています。もし「n」であれば、飛び先へ戻り、初めからやりなおします。

5090行からが修正です。氏名、電話番号とも修正できるようになっています。

もし修正がなければ「0」を入力しリターンキーを押します。この場合、該当者の配列AA$での内容は変化しません。つまり、修正しなかったということになりますね。

修正したければ、画面に表示される矢印の右側から正しいデータをキーインし、次にリターンキーを押します。すると、この修正データが配列AA$とBB$の該当する位置に代入され、前のデータは消えてしまいます。ここでの手助けとなるのが配列X(N)です。この配列に、何番目に代入したらよいかその数値がしまわれていましたね。修正が終ったらRETURNでジャンプしたところへ戻ります。

しかし、まだデータファイルには書き込まれていませんよ。配列の中で修正された、というだけです。

次に、6110行から6180行までは不要ですから削除し、新たに6110行をRETURN文にします。

DELETE

```
delete 6110-6180 ←―――――キーイン後リターンキーを押す
Ok
█
```

```
6000 '
6010 *DAT.SELECT ←―――――ラベルを変更
6020 '
6030 CLS
6040 LOCATE 8,3
6050 PRINT USING "ト゛ウセイ ノ ヒト カ゛ ##メイ イマス";N
6060 FOR I=1 TO N
6070 LOCATE 8,4+I                    (スペース14個)
6080 PRINT USING "## &              &";I,AC$(I)
6090 NEXT I
6100 LOCATE 10,5+I:INPUT "input number";N   ← MをNに変更しました
6110 RETURN ←―――――キーインする
```

この部分は、同姓の人が複数人いた場合、どの人のデータを修正するのか決定するサブルーチンです。

次に7000行からのキー入力待ちのサブルーチンを削除します。

```
delete 7000-7040
Ok
■
```

データ書き込み

そして新たに次のステートメントをキーインします。

```
7000 '
7010 *DAT.WRITE
7020 '
7030 OPEN "2:TELNO1.dat" FOR OUTPUT AS #1
7040 FOR I=1 TO C
7050 WRITE #1,AA$(I),BB$(I)
7060 NEXT I
7070 CLOSE #1
7080 RETURN
```

書き込みモードでファイルをOPEN（7030行）

データの書き込み（7050行）

これは一部修正された配列AA$、BB$に格納されているデータを、あらためて書き込むサブルーチンです。

これでもうおわかりいただけたと思いますが、シーケンシャルデータファイルでの修正は、一度既存のデータを配列内に読み込み、修正があれば、その配列の中味をなおし、正された配列のデータを再びデータファイルに書き込むのです。

ファイル名は同じでも、いままでのデータはすべて消され、まったく新しく書き込まれます。

これでデータ修正のプログラムは完成しましたが、1つだけ書き添えておきます。

それは、このプログラム中でAC$、BC$、Xという名の配列を使用していますが、配列宣言をどこでもしていないことです。このプログラムでは同姓の人がそんなにたくさんいるとは思われませんので、配列宣言をしませんでしたが、このプログラムを何か違うテーマに応用したりして、類似項目が10個以上あるような場合には当然、次のようなステートメントを入れておかなければなりません。

```
115 DIM AC$(20),BC$(20),X(20)
```

次にプログラムリストをかかげます。

```
10 ' file name ==> TELNO3.bas
20 ' 1984.1.25
30 ' sequential data correct
40 ' save "TELNO3.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
120 CLS
130 GOSUB *DAT.READ1
140 DIM AA$(C),BB$(C)
150 GOSUB *DAT.READ2
160 '
170 LOCATE 10,7:PRINT "ﾃﾞｰﾀ ﾉ ｼｭｳｾｲ ｦ ｼﾏｽ"
180 FOR I=1 TO 5000:NEXT I
190 '
200 CLS
210 LOCATE 8,5:PRINT "ﾄﾞﾅﾀ ﾉ ﾃﾞｰﾀ ｦ ｼｭｳｾｲ ｼﾏｽｶ"
220 LOCATE 8,6:PRINT "    input family name"
230 LOCATE 8,7:INPUT "==>";NM$
240 L=LEN(NM$)
250 '
260 GOSUB *DAT.SEARCH
270 IF N=0 THEN GOSUB *N.FOUND:GOTO 200
280 IF N=1 THEN GOSUB *DAT.CORRECT:GOTO 310
290 GOSUB *DAT.SELECT:GOSUB *DAT.CORRECT
300 '
310 CLS
320 LOCATE 12,5:PRINT "ｼｭｳｾｲ ｦ ﾂﾂﾞｹﾏｽｶ"
330 LOCATE 12,6:INPUT "   ( y/n )";ANS$
340 IF ANS$="Y" OR ANS$="y" THEN N=0:GOTO 200
345 GOSUB *DAT.WRITE
350 LOCATE 12,8:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 12,9:INPUT "   ( y/n )";ANS$
370 IF ANS$="y" THEN RUN "menu02.bas"
380 END
1000 '
1010 *DAT.READ1
1020 '
1030 OPEN "2:TELNO1.dat" FOR INPUT AS #1
1040 WHILE NOT(EOF(1))
1050 INPUT #1,A$,B$
```

```
1060 C=C+1
1070 WEND
1080 CLOSE #1
1090 RETURN
2000 '
2010 *DAT.READ2
2020 '
2030 OPEN "2:TELNO1.dat" FOR INPUT AS #1
2040 FOR I=1 TO C
2050   INPUT #1,AA$(I),BB$(I)
2060 NEXT I
2070 CLOSE #1
2080 RETURN
3000 '
3010 *DAT.SEARCH
3020 '
3030 FOR I=1 TO C
3040 IF NM$=LEFT$(AA$(I),L) THEN N=N+1
      :X(N)=I:AC$(N)=AA$(I):BC$(N)=BB$(I)
3050 NEXT I
3060 RETURN
4000 '
4010 *N.FOUND
4020 '
4030 CLS
4040 BEEP 1:FOR I=1 TO 2000:NEXT I:BEEP 0
4050 LOCATE 8,5:PRINT "ｿﾉ ｶﾀ ﾊ ﾄｳﾛｸ ｼﾃｱﾘﾏｾﾝ"
4060 LOCATE 8,7
4070 PRINT "  Start from the beginning!"
4080 FOR I=1 TO 5000:NEXT I
4090 RETURN
5000 '
5010 *DAT.CORRECT
5020 '
5030 CLS
5040 LOCATE 5,2:PRINT AC$(N);" ｻﾝ ﾉ ｼｭｳｾｲ"
5050 LOCATE 5,3
5060 INPUT "    ﾏﾁｶﾞｲﾅｲﾃﾞｽｶ ( y/n )";ANS$
5070 IF ANS$="N" OR ANS$="n" THEN RETURN
5080 LOCATE 5,4
5090 PRINT "ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ(ｼｭｳｾｲ ﾅｼ=0)"
5100 LOCATE 5,5:PRINT "ｼﾒｲ= ";AC$(N)
5110 LOCATE 5,6:INPUT "    ==> ";NM$
```

```
5120 IF NM$<>"0" THEN AA$(X(N))=NM$
5130 LOCATE 5,7:PRINT "TEL No.= ";BC$(N)
5140 LOCATE 5,8:INPUT "    ==>";TEL$
5150 IF TEL$<>"0" THEN BB$(X(N))=TEL$
5160 RETURN
6000 '
6010 *DAT.SELECT
6020 '
6030 CLS
6040 LOCATE 8,3
6050 PRINT USING "ト゛ウセイ ノ ヒト カ゛ ##メイ イマス";N
6060 FOR I=1 TO N
6070 LOCATE 8,4+I
6080 PRINT USING "## &                 &";I,AC$(I)
6090 NEXT I
6100 LOCATE 10,5+I:INPUT "input number";N
6110 RETURN
7000 '
7010 *DAT.WRITE
7020 '
7030 OPEN "2:TELNO1.dat" FOR OUTPUT AS #1
7040 FOR I=1 TO C
7050 WRITE #1,AA$(I),BB$(I)
7060 NEXT I
7070 CLOSE #1
7080 RETURN
```

いかがですか？　バグはありませんでしたか？

それではさっそく実行してみましょう。このまま f·5 キーを押してもかまいませんし、メニュープログラムをロードして、選択枝でこのデータ修正の仕事を選んでもかまいません。いずれにしても正常に動くはずです。

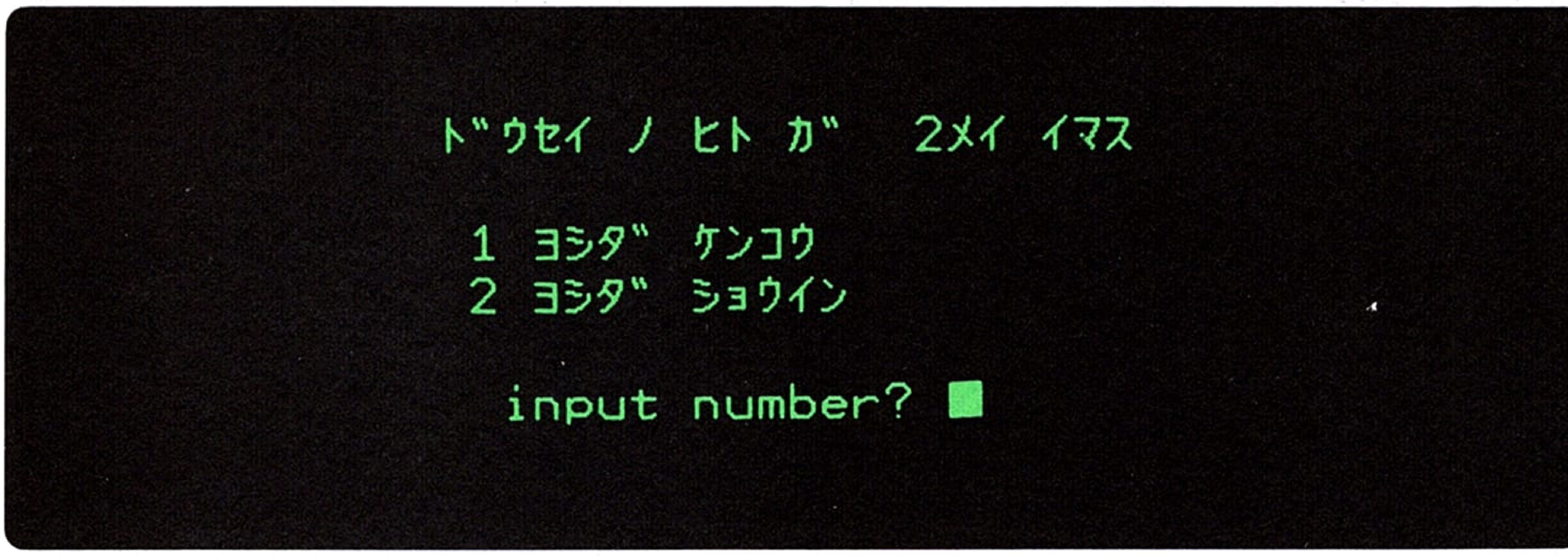
●プログラムの間違いのことをバグ（虫）といいます。

最初に次の画面となります。

```
ト゛ナタ ノ テ゛ータ ヲ シュウセイ シマスカ
   input family name
==>? ■
```

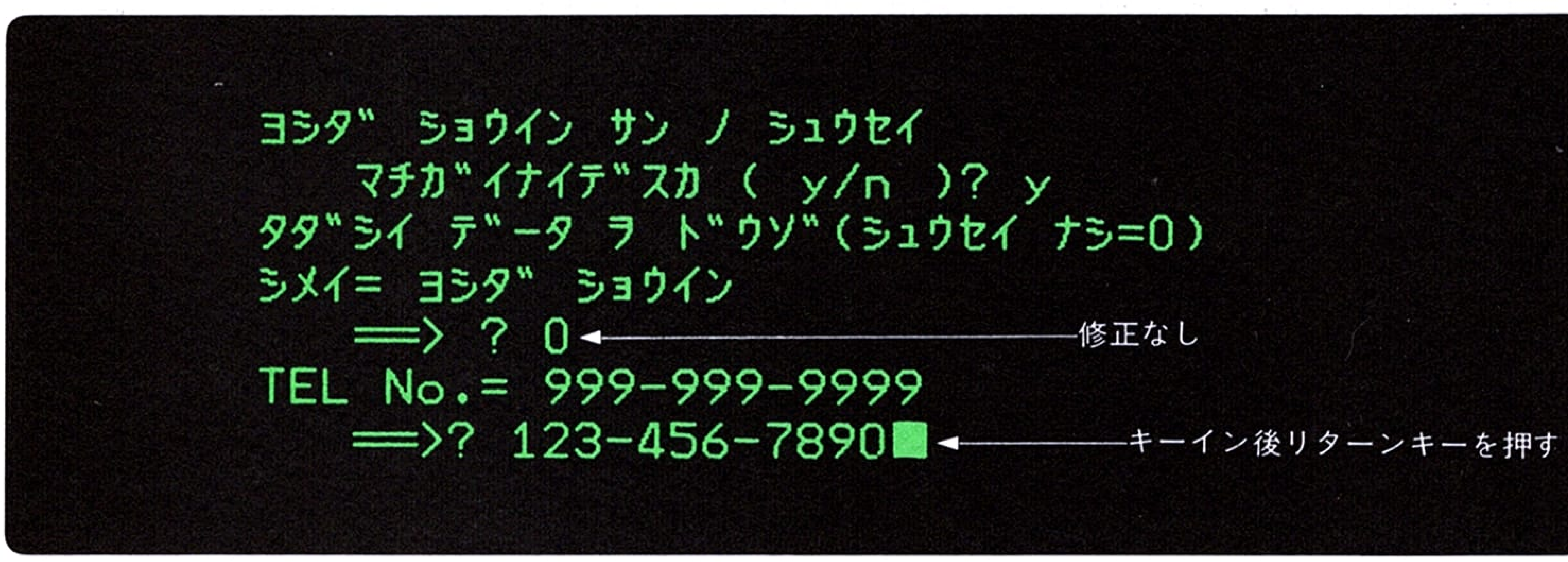
ここでは「ヨシダ」と入力してみましょう。

するとすぐに配列AA$の中から「ヨシダ」という姓を探し出し、画面に表示します。どうやら同姓の人が2人いたようです。



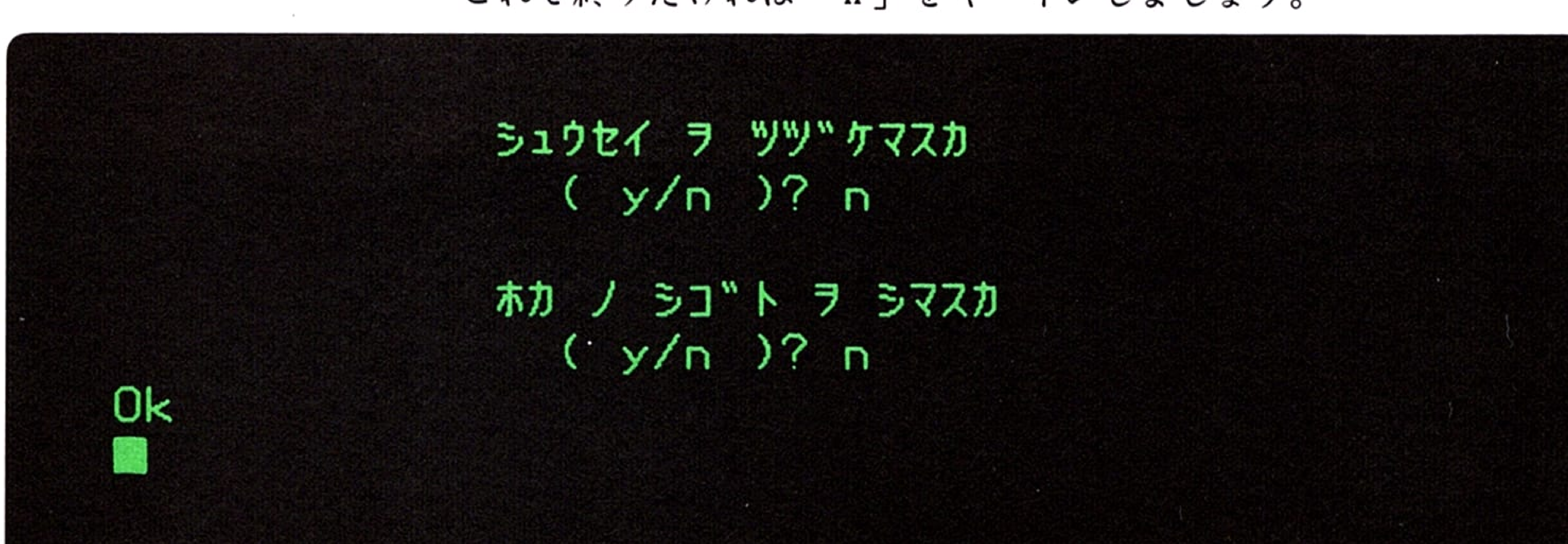
2番目の「ヨシダ」さんを選んでみましょう。

2 ⏎ とキーインして次の画面です。



氏名は修正しないことにして0をキーインしてからリターンキー。電話番号を変更してみました。キーインしてリターンキーを押すと、次の画面になり、「修正を続けますか」の問いかけに「n」を入力すると、修正データがデータファイルに書き込まれ、他の仕事をするかどうかのメッセージが現れます。

これで終りたければ「n」をキーインしましょう。



3·7 データを印刷する

私たちはこれまで電話番号データファイルの作成をし、データの追加をしたり修正をしたり、また希望する人の電話番号を知ることにも成功しました。

しかし、これらはすべてディスプレイ上で行ってきましたし、登録データすべてをひと目で見るというわけにはまいりませんでした。

また、データファイル作成中は、プログラムに入力ミスがないかどうかのルーチンもつけてはいますが、キー操作は人間のやることです、けっこう間違いがあるものです。

この節では、登録データをプリンタで印刷し、もう一度この目で確かめてみましょう。

また、いままではディスクのデータを読み出して画面に表示はできましたが、それでもまだ、ディスクにほんとうに書き込まれているという実感がなかなかもてませんでした。しかし、プリンタで実際に印刷されてしまえば、信じざるをえないというものです。

画面にデータを表示

登録してあるデータを印刷する前に、まずディスプレイ画面に表示してみましょう。というのは、画面に表示するにはPRINT文で、一方、プリンタのほうに出力するには LPRINT 文を使えばよいわけですから、画面表示さえできていれば、あとはPRINTの前にLをつけるだけですから、ほぼ出来上がったも同然だからです。それに、ディスプレイ上のほうが操作しやすいですね。

PRINT
LPRINT

それでは前節のプログラム「TELNO3.bas」をロードしてください。

次に、130行から380行まで、3000行から7080行まで削除してください。3000行からは不要ですし、130行からは作りなおさなければなりません。

```
delete 130-380
Ok
delete 3000-7080
Ok
■
```

不要な行を削除

次に、最初の部分にファイル名など多少の変更を加えます。

```
10 ' file name ==> TELNO5.bas
20 ' 1984.1.25
30 ' sequential data print out
40 ' save "TELNO5.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 80,20
110 DEFINT A-Z
120 CLS
130 '
```

新ファイル名（10行・40行の TELNO5.bas）

横の桁数を80桁に変更（100行の 80）

氏名と電話番号を横に続けて並べますので、これまでの40桁ではちょっときゅうくつです。80桁にしておきましょう。

画面表示までのプロセスですが、まず、1000行からのサブルーチンへデータを読みに行き、ここでデータをカウントして配列を宣言します。次に2000行からのサブルーチンでデータをいま宣言した配列に読み込み、これをディスプレイ上に表示すればよいわけです。

メイン部分は次のように作りました。

```
130 '
140 *MAIN
150 '
160 LOCATE 25,3:PRINT "■|   ﾃﾞｰﾀ ﾉ ｲﾝｻﾂ   |■"
170 LOCATE 27,4:PRINT "5 ﾒｲｽﾞﾂ ﾋｮｳｼﾞ ｼﾏｽ"
180 GOSUB *DAT.READ1
190 DIM AA$(C+5),BB$(C+5)
200 GOSUB *DAT.READ2
210 GOSUB *DAT.DISPLAY
220 LOCATE 25,4:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
230 LOCATE 25,5:INPUT "     ( y/n )";ANS$
240 IF ANS$="y" THEN RUN "menu02.bas"
250 END
```

180行：配列設定のためのデータ数読み出し

200行：配列にデータを読み込む

210行：画面表示のサブルーチンへ

190行の配列宣言で、その大きさを従来とちがってプラス５していますが、これはデータをディスプレイに表示をする場合、５つずつを１ブロックとしていますので、このようにしました。

というのは、データがいつも５で割り切れる数、たとえば30とか45とかで終っていればよいのですが、はんぱな数18とか47とかで終っていた場合、画面表示をする際、配列AA$、BB$に、この数を超える添え字が入りますので、次のエラーとなってしまうからです。

```
Subscript out of range in 3070
```

次に画面表示部分のプログラムを示します。

これもサブルーチンにしました。

```
3000 '
3010 *DAT.DISPLAY
3020 '
3030 FOR I=1 TO C STEP 5
3040  FOR J=1 TO 5
3050  M=I+J-1
3060  LOCATE 15,5+J
3070  PRINT USING "## &               &";M,
      AA$(M);
3080  PRINT USING " TEL  &              &";
      BB$(M)
3090  NEXT J
3100  LOCATE 25,12:PRINT "Hit RET key for next"
3110  Y$=INKEY$:IF Y$="" THEN GOTO 3110
3120 NEXT I
3130 CLS
3140 RETURN
```

配列の添字を計算（3050行）

15スペース（3070行）

14スペース（3080行）

何かキーが押されると次の行を実行（3110行）

変数JのFOR～NEXTループで5名の氏名と電話番号を表示し、変数IのFOR～NEXTループで次のパターンへ移ります。

STEP

5名ずつの表示ですから変数Iのきざみは5、STEP 5をつければよいですね。この場合Iは、1、6、11、……というように5きざみで変化していきます。そしてこの間の落ちている数、2、3、4、5、7、8、9、10、……などは、ループ変数Jの値を加えて調整し、1、2、3、4、5、6、7、……、10、11、……という連続した整数列を作り出しています。この値は3050行で計算され、Mに代入されます。このMの値を配列変数AA$、BB$の添字に使うわけですね。

3110行はキー入力待ちで、何かキーが押されるまでは、この行でとどまっています。

つまり、どんなキーでもよいですから、キー入力があるとこの行を抜けて3120行のNEXT文に行き、次の5名が表示されるわけです。

PRINT USING

画面の表示には、前に説明したPRINT USING文を使っています。

&と&の間のスペースの数は、氏名ではなく住所などですともっとふやさなければならないでしょう。スペースの数とか、表示する位置、つまりLOCATE文の座標などは、実際にプログラムを実行させて、調整するとよいでしょう。

さあ、これで完成です。1000行および2000行からのサブルーチンは削除しませんでしたから、ちゃんと残っていますね。リストコマンドで確認してみてください。

だいじょうぶでしたら、**このプログラムをセーブ**しておきましょう。

```
save "TELNO5.bas" ⏎
```

ここで、さっそく実行してみたいのですが、その前にもう少しデータを入れておきましょう。

メニュープログラムをロードしてRUNしてください。

```
run "menu02.bas" ⏎
```

メニュー画面が現れましたら、2番の「TEL No ノ ツイカ」を選択します。

画面のメッセージに従って、次のデータを入力してください。

```
ｻﾙﾄﾋﾞ ｻｽｹ･････････000-111-2222
ｷﾘｶﾞｸﾚ ｻｲｿﾞｳ･･････333-444-5555
ﾐﾔﾓﾄ ﾑｻｼ･･････････666-777-8888
```

入力が終りましたら、「ホカ ノ シゴト ヲ シマスカ」のメッセージに対して「y」をキーインし、メニュー画面に戻ります。

そこで5番の「TEL LIST ノ インサツ」を選びます。

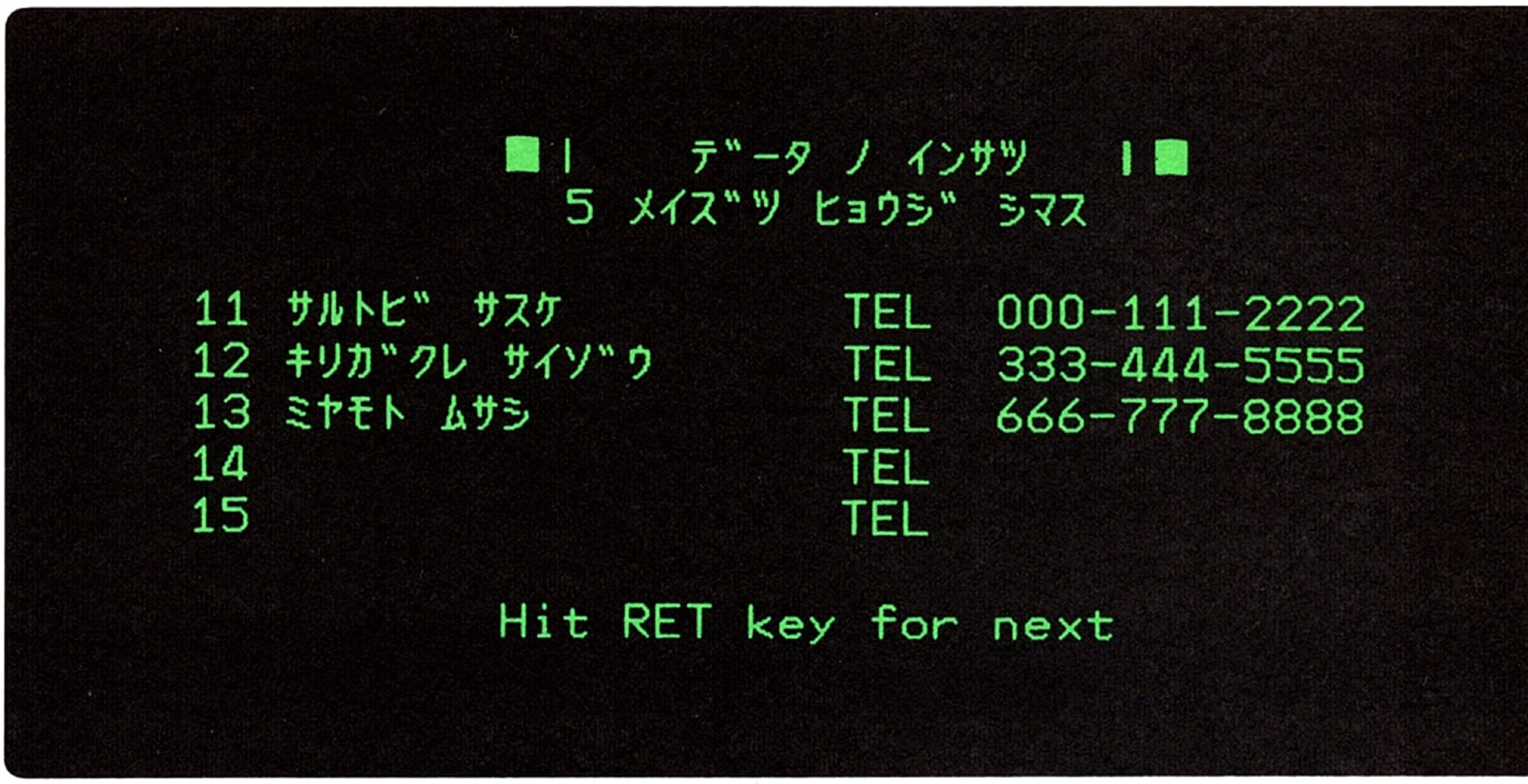
やがて、このプログラムが実行され、次の画面が現れます。

```
        ■|    ﾃﾞｰﾀ ﾉ ｲﾝｻﾂ    |■
          5 ﾒｲｽﾞﾂ ﾋｮｳｼﾞ ｼﾏｽ

  1 ﾄｸｶﾞﾜ ｲｴﾔｽ          TEL  000-000-0000
  2 ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ          TEL  111-111-1111
  3 ｵﾀﾞ ﾉﾌﾞﾅｶﾞ          TEL  222-222-2222
  4 ｱｹﾁ ﾐﾂﾋﾃﾞ           TEL  333-333-3333
  5 ｱﾗｲ ﾊｸｾｷ            TEL  444-444-4444

        Hit RET key for next
```

これは、データファイルの中の最初の5名が表示されたところです。ここで ⏎ キーを押しますと次の5名、もう一度押すと先ほど書き込んだ3名の電話番号が表示されます。



どうですか？　ちゃんと書き込まれていましたか？

ここでリターンキーを押しますと、ほかの仕事をするかどうかのメッセージが表示されます。とりあえず「n」とキーインしましょう。

データをプリンタに出力

これで一応データを画面に表示することはできました。次はプリンタに出力して印刷してみましょう。

この操作は、ここまでできていればきわめて簡単で、画面表示のためのPRINT文をLPRINT文に変えるだけでいいのです。しかし、せっかく画面に表示したのですから、これはそのまま生かし、さらにLPRINT文を追加することにしましょう。そうすれば、画面に一度表示し、それを印刷できるという一石二鳥の効果があります。

LPRINT文の入れる場所ですが、PRINT文のすぐうしろ、3080行と3090行の間に挿入すればよいでしょう。

次のステートメントを追加しますが、これはスクリーンエディタによって簡単にできます。わざわざキーインする必要はありません。

```
3085  LPRINT USING "## &                    &";M,
       AA$(M);
3086  PRINT USING " TEL  &                   &";
      BB$(M)
```

まず、3070行の3の上にカーソルを重ね、[3] [0] [8] [5]とキーインし、次にPRINTのPの上にカーソルを重ね (INS) キーを押し、インサートモードにしてからLをキーインし [↵] キーを押して終りです。

3086行は同じように3080行を修正すればよいですね。

このスクリーンエディタによって、3080行、3090行は消えることはありませんからご心配なく。

行番号が少しふぞろいになりました。3085行以降を10番きざみになおしておきましょう。

```
renum 3090,3085
```

これで完成です。まずプログラムをセーブしてください。

そして、RUNさせてみましょう。次のように打ち出されてきましたか?

```
 1 トクガ゛ワ イエヤス        TEL  000-000-0000
 2 トヨトミ ヒデ゛ヨシ        TEL  111-111-1111
 3 オダ゛ ノブ゛ナガ゛        TEL  222-222-2222
 4 アケチ ミツヒデ゛          TEL  333-333-3333
 5 アライ ハクセキ            TEL  444-444-4444
 6 カイハ゛ラ エッケン        TEL  555-555-5555
 7 カイホ セイリョウ          TEL  666-666-6666
 8 ヨシダ゛ ケンコウ          TEL  777-777-7777
 9 サトウ イッサイ            TEL  888-888-8888
10 ヨシダ゛ ショウイン        TEL  123-456-7890
11 サルトビ゛ サスケ          TEL  000-111-2222
12 キリガ゛クレ サイソ゛ウ    TEL  333-444-5555
13 ミヤモト ムサシ            TEL  666-777-8888
14                            TEL
15                            TEL
```

50音順に印刷する

前項で電話番号のリストを無事印刷することができましたが、これは私たちがキーボードから入力した登録した順になっていて、氏名が50音順になっていません。これを50音順に並べかえて印刷すれば、何かと使い勝手もよくなるのではないかと思います。

ソート

このように順序を並べかえることをソートと呼び、ビジネスの分野では、数値を大きい順に並べたり、小さい順に並べたり、とにかく、よく使われます。

私たちも、ここでソートにチャレンジしてみましょう。

この電話帳リストでのソートは、数値ではなく文字を並べかえることになりますが、原理は同じですので、また、そのほうが理解しやすいと思いますので、数値を例にとって説明します。

次に簡単なソートプログラムをかかげます。

〈注〉下のプログラムは説明のためですからキーインしなくてもかまいません。キーインする場合はNEWコマンドを実行してから入力してください。

```
100 ' sorting
110 '
120 FOR I=1 TO 5
130  READ A(I)               ← ソートするデータを配列に読み込む
140  PRINT A(I)"   ";        ← ソート前のデータを画面に表示
150 NEXT I
160 PRINT                    ← これを入れておかないとソート後のデータが続いて表示されてしまう
170 '
180 FOR I=1 TO 4
190 K=I
200  FOR J=I+1 TO 5
210  IF A(J)<A(K) THEN K=J   ← 小さい値があればポインタKをその値のところに合わせる
220  NEXT J
230  SWAP A(I),A(K)          ← ポインタKの指し示す配列A(K)の中味と配列A(I)の中味を入れかえる
240 NEXT I
250 FOR I=1 TO 5
260  PRINT A(I)"   ";        ← ソート後のデータを画面に表示
270 NEXT I
280 '
290 DATA 4,1,5,3,2           ← ソートするデータ
```

このソートはバブルソートと呼ばれるもので、もっとも基本的なものです。

方法は、まずソートするデータを配列に読み込み、ある１つのデータを基準にして、次々と他のデータと大きさを比較していきます。

たとえば、小さい順に並べるのでしたら、基準となっている数より小さい数が現れたら、今度はその数を基準にし、次のデータと比較していきます。

このような操作を次々とくりかえし、最後に基準になっているデータを先頭のほうへもってきます。

ポインタ

この比較する基準となるデータは、ポインタによって指し示されています。上の例でいいますと、Kがポインタになっています。

SWAP

小さいものを先頭にもってくる作業はSWAP文で行います。

230行のSWAP文の意味は、A(I)とA(K)の中味を入れかえるということです。上のプログラムにそって、もう少し説明を加えます。

最初のループ、つまりI＝1でJのループに入ったときの状態は次のようになっています。K＝1ですからポインタは矢印の所にあります。

A(1)	A(2)	A(3)	A(4)	A(5)
4	1	5	3	2
↑				

ここでA(1)とA(2)の中味を比較します。4＞1、つまりポインタが指し示すA(1)の中味より小さい値がでてきたわけですから、ポインタをその小さいほうの値のところに合わせます。この場合2番目ですからK＝2になります（210行のK＝Jの操作をしたわけです）。ポインタが移動して次のような状態になりました。

A(1)	A(2)	A(3)	A(4)	A(5)
4	1	5	3	2
	↑			

次に、ポインタが指し示すA(2)の中味とA(3)の中味を比較します。この場合は1＜5ですからポインタは動きません。

次に、やはりポインタが指し示すA(2)の中味とA(4)の中味を比較します。この場合も1＜3ですから、ポインタは相変わらずA(2)のところを指し示しています。

最後、現在のポインタ位置A(2)の中味とA(5)の中味を比較します。このときの状態は次のようになっています。

A(1)	A(2)	A(3)	A(4)	A(5)
4	1	5	3	2
	↑			

当然1＜2ですから、ポインタの位置は変わりません。

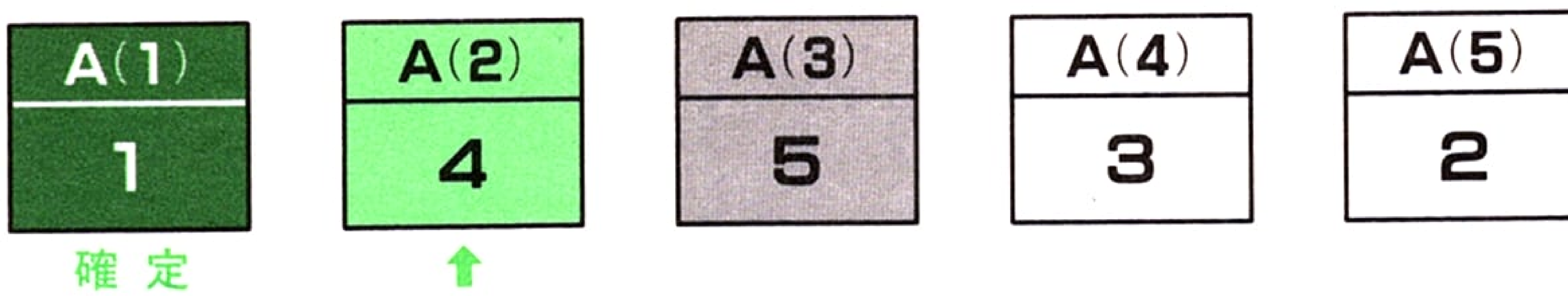
これでI＝1のときのJのループを1回終えたわけです。ですから次のIのループに行く前に、最小値、つまりポインタが指し示している配列の中味をA(1)の中味と入れかえなければなりません。この操作を230行のSWAP文でやっています。

入れかえた状態は次のようになり、A(1)の中味は5つの数の中の最小値と確定しました。

A(1)	A(2)	A(3)	A(4)	A(5)
1	4	5	3	2

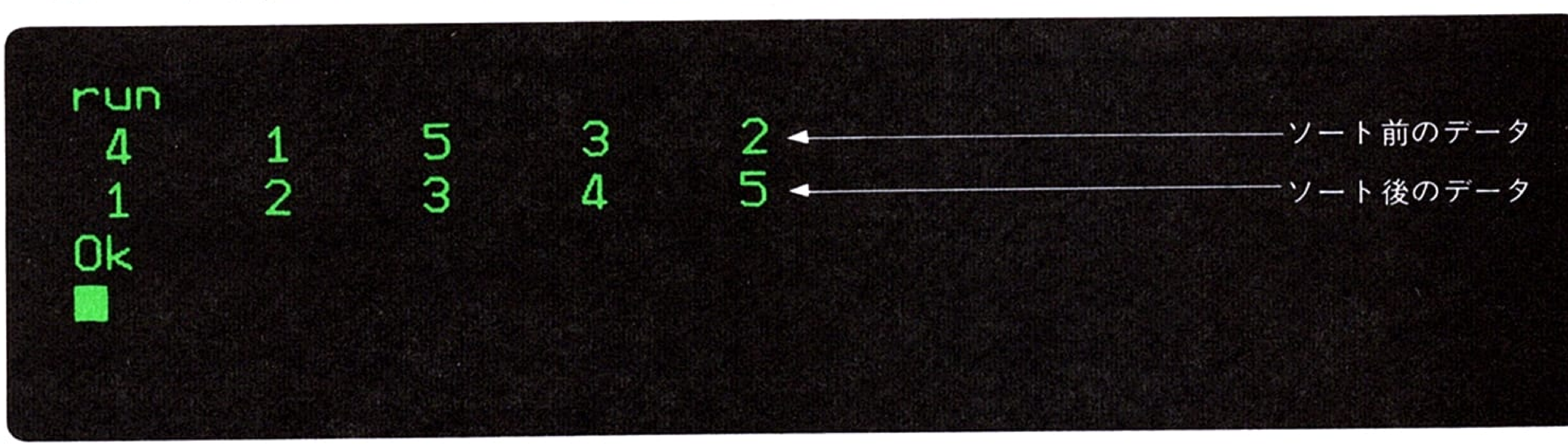
次にI＝2のループでは、最初のポインタKはK＝Iですから2にセットされ、図のような状態になり、ポインタの指し示すA(2)の中味とA(3)の中味を比較することから始まります。

以下、I＝1のループの説明と同じ操作がくりかえされます。



このループでは、ポインタは最終的にはA(5)を指し示すことになり、A(2)の中味とA(5)の中味がSWAP文により入れかわります。

105ページのサンプルプログラムを実行すると次のようになります。

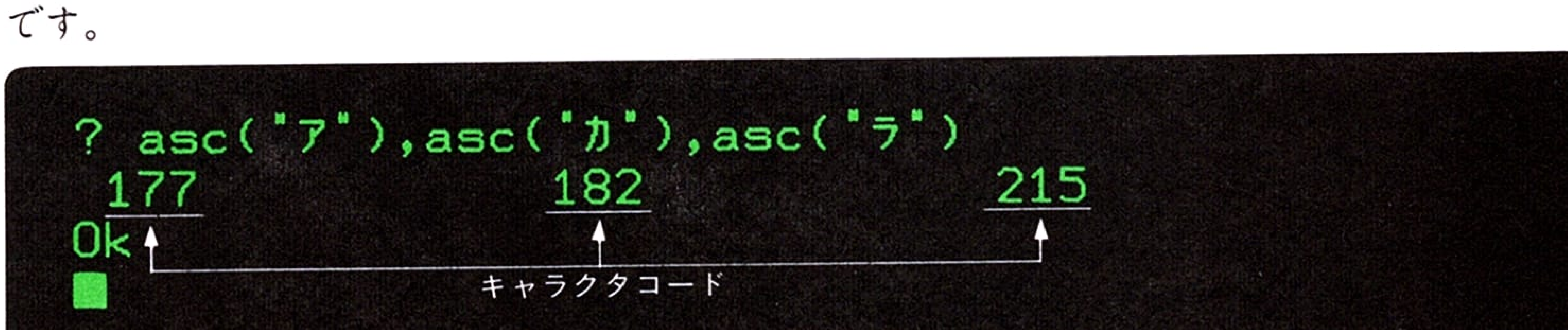


ソートの原理はわかっていただけましたか？

次に本項のテーマ、氏名を50音順に並べかえるソートに入りますが、文字であっても仕組みはまったくかわりません。ただ、文字の場合にはASC関数を使います。この関数は、文字を対応するキャラクタコードに変換する機能があります。

したがって、ソートするにはキャラクタコードの大小によって行います。

ユーザーズマニュアルのキャラクタコード表を見ればわかりますように、「ア」が&HB1、「ン」が&HDD、10進数になおして177から221までとなっています。この数字の大小で、どちらの文字が先にあるかわかるわけです。つまり、文字のソートも数字のソートに帰するわけです。



ソーティング部分もサブルーチンにします。プログラムを次に示します。

```
4000 '
4010 *SORT
4020 '
4030 FOR I=1 TO C-1
4040 K=I
4050  FOR J=I+1 TO C
4060   IF ASC(LEFT$(AA$(J),1))<ASC(LEFT$(AA$
(K),1)) THEN K=J
4070  NEXT J
4080  SWAP AA$(I),AA$(K):SWAP BB$(I),BB$(K)
4090 NEXT I
4100 RETURN
```

電話番号も同時に入れかえる

〈注〉ソートのサンプルプログラムを入力した人は「TELENO5.bas」をロードしてからソーティングサブルーチンをキーインしてください。

105ページのサンプルプログラムと比べていただくと、形がだいたい同じですから、わかると思います。

ただ、SWAP文で氏名を入れかえるとき、電話番号も同時に入れかえることを忘れないでください。

LEFT$

4060行のLEFT$文は、配列AA$(J)、AA$(K)の一番左の1文字をそれぞれ取り出し、それをASC関数でキャラクタコードに変換して比較しています。つまり、2文字目のソートは行っておりませんので、1文字目だけが50音順に整理されることになります。

2文字目のソートも同じようにできますが、データ量が少なければそれほど必要もないでしょう。

次に、印刷形式に2つのオプションを持たせることにします。つまり、登録順と50音順に印刷する方法です。このためのステートメントを170と180行の間に入れておきます。

```
172 LOCATE 30,6:PRINT "1 ﾄｳﾛｸ ｼﾞｭﾝ"
174 LOCATE 30,7:PRINT "2 50ｵﾝ ｼﾞｭﾝ"
176 LOCATE 28,9:INPUT "input number";P
178 IF P=2 THEN FLAG=1
```

2を選択したらフラグを立てる

フラグのセット

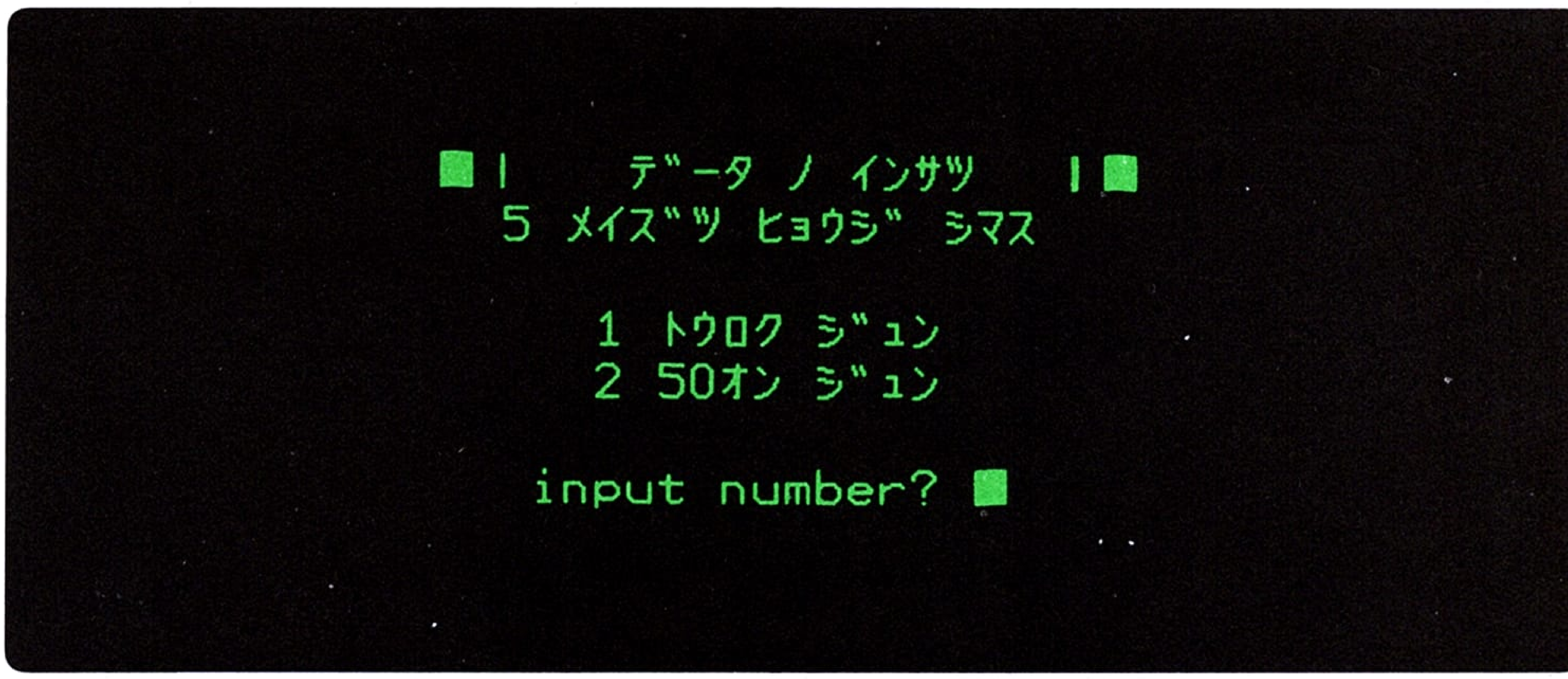
176行のINPUT文で2番を選択した場合に、178行でフラグをセット(1にする)しています。このフラグによって、どちらを選んだかわかるわけです。次のステートメントを追加します。

```
205 IF FLAG=1 THEN GOSUB *SORT
```

フラグが1にセットされていれば、ソーティングサブルーチンへ飛ぶことになります。

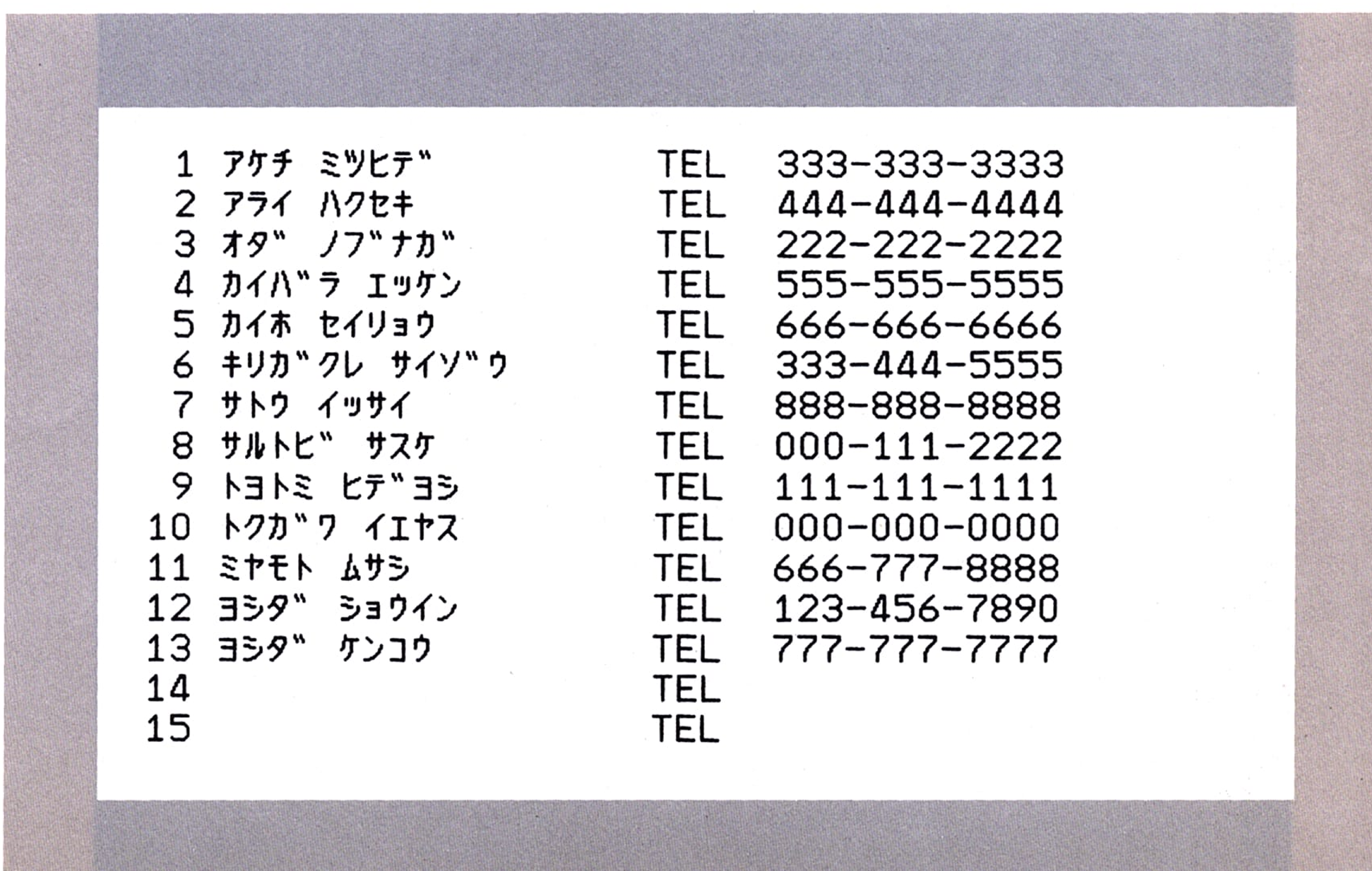
ソートする、しないにかかわらず、印刷部分はまったく同じです。

やっと、この節のプログラミングも終りました。実行させてみましょう。［f･5］キーを押すと次の画面になります。



ここでは2をキーインし、50音順に印刷してみましょう。

いかがですか？　ソートされていますか？



このプログラムのセーブはもうしましたか？　忘れないようにしてください。

次にプログラムリストをのせておきます。

```
10 ' file name ==> TELNO5.bas
20 ' 1984.1.25
30 ' sequential data print out
40 ' save "TELNO5.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 80,20
110 DEFINT A-Z
120 CLS
130 '
140 *MAIN
150 '
160 LOCATE 25,3:PRINT "■|    ﾃﾞｰﾀ ﾉ ｲﾝｻﾂ    |■"
170 LOCATE 27,4:PRINT "5 ﾒｲｽﾞﾂ ﾋｮｳｼﾞ ｼﾏｽ"
172 LOCATE 30,6:PRINT "1 ﾄｳﾛｸ ｼﾞｭﾝ"
174 LOCATE 30,7:PRINT "2 50ｵﾝ ｼﾞｭﾝ"
176 LOCATE 28,9:INPUT "input number";P
178 IF P=2 THEN FLAG=1
180 GOSUB *DAT.READ1
190 DIM AA$(C+5),BB$(C+5)
200 GOSUB *DAT.READ2
205 IF FLAG=1 THEN GOSUB *SORT
210 GOSUB *DAT.DISPLAY
220 LOCATE 25,4:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
230 LOCATE 25,5:INPUT "     ( y/n )";ANS$
240 IF ANS$="y" THEN RUN "menu02.bas"
250 END
1000 '
1010 *DAT.READ1
1020 '
1030 OPEN "2:TELNO1.dat" FOR INPUT AS #1
1040 WHILE NOT(EOF(1))
1050 INPUT #1,A$,B$
1060 C=C+1
1070 WEND
1080 CLOSE #1
1090 RETURN
2000 '
2010 *DAT.READ2
```

```
2020 '
2030 OPEN "2:TELNO1.dat" FOR INPUT AS #1
2040 FOR I=1 TO C
2050   INPUT #1,AA$(I),BB$(I)
2060 NEXT I
2070 CLOSE #1
2080 RETURN
3000 '
3010 *DAT.DISPLAY
3020 '
3030 FOR I=1 TO C STEP 5
3040  FOR J=1 TO 5
3050  M=I+J-1
3060  LOCATE 15,5+J
3070  PRINT USING "## &                    &";M,
     AA$(M);
3080  PRINT USING " TEL  &                    &";
     BB$(M)
3090  LPRINT USING "## &                    &";M,
      AA$(M);
3100  LPRINT USING " TEL  &                    &";
      BB$(M)
3110  NEXT J
3120  LOCATE 25,12:PRINT "Hit RET key for next"
3130  Y$=INKEY$:IF Y$="" THEN GOTO 3130
3140 NEXT I
3150 CLS
3160 RETURN
4000 '
4010 *SORT
4020 '
4030 FOR I=1 TO C-1
4040 K=I
4050  FOR J=I+1 TO C
4060   IF ASC(LEFT$(AA$(J),1))<ASC(LEFT$(AA$
(K),1)) THEN K=J
4070  NEXT J
4080  SWAP AA$(I),AA$(K):SWAP BB$(I),BB$(K)
4090 NEXT I
4100 RETURN
```

第4章

ランダムファイル[I]



4

4・1　ランダムファイルについて

ランダムファイルとはいったいなんでしょうか。

私たちが前章までに勉強したシーケンシャルファイルは、あるデータを読み出したいといっても直接読み出すことはできませんでした。検索にしろ、修正にしろ、そのデータファイルに書き込まれているデータを、好き嫌いにかかわらず、初めから読み込んでいかなければなりませんでした。

要するに、長さのまちまちのデータが連続的に書き込まれていたわけですから、書き込まれたデータの並びには規則性がありません。

これに対してランダムファイルは、データが一定の決められた長さで書き込まれていますから、当然、データの書き込まれ方に規則性がでてきますので、どこに位置しているかということさえわかれば、直接に読み出すことができる

右ページの図をごらんください。

これは、前章の電話帳の氏名・電話番号のシーケンシャルファイルとランダムファイルでの書き込まれ方を図式化したものです。

シーケンシャルファイルでは、各項目が、コンマで区切られているだけで、ビッシリつめこまれています。△がありますが、これは姓と名を分けるために必要なスペースです。

文字は1文字が1バイトで書き込まれますから（ただし濁点は1文字分ですから「ガ」などは2バイト必要ですが、説明上ここでは1バイトと考えます)、「トクガワ　イエヤス」は9バイト、「オダ　ノブナガ」は7バイトというように、長さは一定しておりません。

可変長レコード

このようにシーケンシャルファイルではレコードの長さが決まっていませんので「可変長レコード」と呼ばれ、制限もなく、ディスクの使用効率も非常によいのですが、ファイルの何番目に位置しているかは、一度データを全部読み出してみなければわかりません。

一方ランダムファイルは、図のように氏名および電話番号のうしろにムダなスペース（△印）がたくさんあいておりますが、いかにも整然と並んでいます。

図の場合ですと、どのデータも氏名に17バイトが、電話番号に21バイトが割り当てられています。シーケンシャルファイルでしたら「トクガワ　イエヤス」の場合、氏名と電話番号で9＋12で21バイトですむところをランダムファイルでは17バイトもよけいにムダな使い方をしているのです。

しかし、メリットもおおいにあります。各データの書き込まれる長

トクガワ△イエヤス	,	000−000−0000	,	トヨトミ△ヒデヨシ	,	111−1
11−1111	,	オダ△ノブナガ	,	222−222−2222	,	アケチ△ミツヒデ
	,	333−333−3333	,			
	,	キリガクレ△サイゾウ	,	333−444−		
5555	,	ミヤモト△ムサシ	,	666−777−8888	,	

⬆シーケンシャルファイル

ランダムファイル⬇

トクガワ△イエヤス△△△△△△△△	000−000−0000△△△△△△△△
トヨトミ△ヒデヨシ	111−111−1111
オダ△ノブナガ	222−222−2222
アケチ△ミツヒデ	333−333−3333
キリガクレ△サイゾウ	333−444−5555
ミヤモト△ムサシ	666−777−8888

固定長レコード

さが一定していますので（ランダムファイルのレコードは「固定長」レコードといいます)、インデックスにより、誰々さんは何番目に書き込まれているということがわかれば、シーケンシャルファイルのように全部読み出さなくても直接にその個所だけ読み出すことができます。

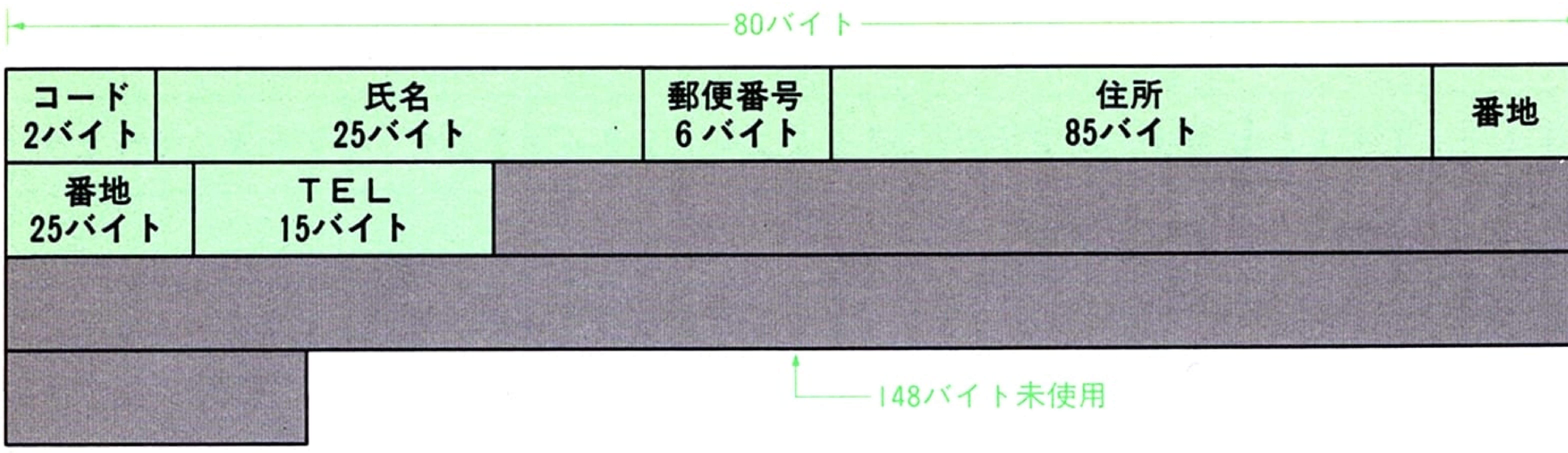
セクタ

1セクタ＝256バイト

第2章第2節で簡単にふれましたが、ディスクにはサーフェス、トラック、セクタという単位がありました。ランダムファイルでの読み書きは、一番小さな単位、つまりセクタごとに行われます。そして1セクタは256バイトです。

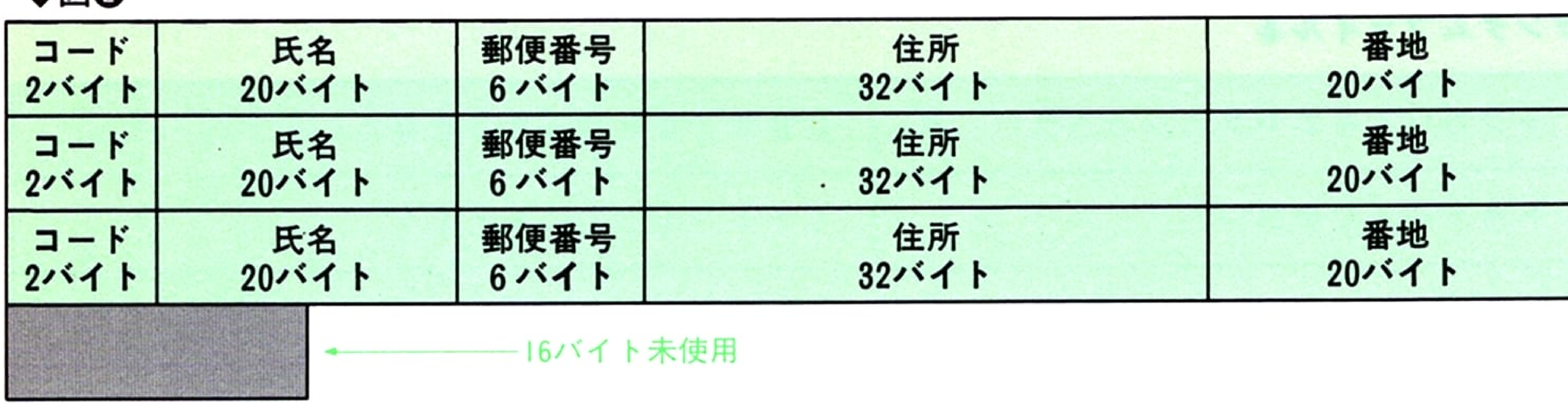
ですから、この256バイト内で各項目に決められた（固定された）長さを割り当てなければなりません。

次ページの図をごらんください。図❶、図❷とも、1セクタ256バイトにDM（ダイレクトメール）用の各項目を割り当てたものですが、図❶には1名分、図❷では3名分入っています。図❶の場合は、ずいぶん未使用の部分が多くムダな使い方をしていますし、一方図❷では



↑図❶

↓図❷



余りが16バイトで大変効率的です。

このくらいの人数ですと、メモリのムダづかいの様子はまだ実感としてわかりませんが、たとえば100人のDM（ダイレクトメール）用データファイルを作る場合、書き込みは256バイト（1セクタ）単位で行われますから、図❶の場合100セクタ必要になってきます。ところが図❷ですと、34セクタあれば十分です。66セクタも差がでてきます。

このように、1セクタ内でフォーマットをどう決めていくかによって、ランダムファイルにおいても、シーケンシャルファイルと同じようにディスクを有効に活用できるのです。

こう考えますと、慣れてしまえば、ランダムファイルのほうがずっと便利に使えますし、いや、**ビジネスの分野ではどうしてもランダムファイルでなければ用をなさない**といったほうがよいかもしれません。

以上、ランダムファイルについての概略を述べましたが、少しわかりにくかったかもしれませんが、わからなければそのままにして、少しずつ具体例を学習していきましょう。やがて、その仕組が理解できると思います。

コンピュータは理屈、理論の面もおおいにありますが、とくにソフトのレベルでは、実際にやってみて、体で覚えてしまうのが一番早いようです。

ファイルをOPENする

ランダムファイルの場合もシーケンシャルファイルと同じように、データファイルに何か書き込んだり、あるいは読み出したりするとき、ファイルをOPENしなければなりません。

これは、前にも説明しましたが、本とかノートの内容を見るときにページを開かなければならないのと同じです。

書式は、シーケンシャルファイルのように「モード」の指定はなく次のようになっています。

OPEN

```
OPEN "2:DRMAIL.dat" AS #1
```

- 2 — ドライブ番号
- DRMAIL.dat — ファイル名
- 1 — ファイル番号

ファイル名の扱いはシーケンシャルファイルのときと同じです。ドライブ1の場合にはドライブ番号は省略できます。

ファイル番号（使用するバッファ番号）は、ディスク起動時の

```
How many files(0-15)? ■
```

で指定した数を超えてはいけません。

また、シーケンシャルファイルでは「モード」の指定がありましたから、INPUTモード、APPENDモードの場合には、OPENしたときディスクにファイルがないと、

```
File not found
```

のエラーメッセージがでましたが、ランダムファイルの場合はOPEN文だけではエラーは出ません。

このOPEN文でファイルを開いてはじめて、データを読んだり書いたりすることが可能になります。

ファイルをCLOSEする

ファイルをOPENして、使い終ったらCLOSEすることにしましょう。これは、シーケンシャルファイルの場合とまったく同じです。

CLOSE

```
CLOSE #1,#2
```

- 1, 2 — ファイル番号

ファイル番号が省略されたときは、そのときOPENしているファイルがすべてCLOSEされます。ファイル番号は上記のように2つでも

3つでもかまいません。コンマで区切って複数指定できます。

データ書き込みの前準備

ファイルをOPENしたということは、「そのファイルをこれから使いますよ」と宣言したことにはなりますが、このままでは書き込みはできません。前準備が必要なのです。

116ページでダイレクトメール用のデータの割り当てを行ったように、1セクタ（256バイト）の中を、どんな項目でどんな割りつけをして使うのか、その形を決めておかなければいけないのです。

氏名には何バイト、住所には何バイト、というように定義しておく必要があるのです。この定義はFIELD文によって行いますが、ランダムファイルでは、この決められたフォーマットに従って、どのレコードも同じ様式で書き込まれます。

FIELD文

FIELD文の書式は次のとおりです。

```
FIELD #1,15 AS A1$,20 AS A2$,50 AS A3$
```

- 文字変数: A1$, A2$, A3$
- フィールド幅（バイト数）: 15, 20, 50
- ファイル番号: 1

この例ですと、ファイル番号1番（というより1番バッファ）に文字変数A1＄、A2＄、A3＄にそれぞれ15バイト、20バイト、50バイトのフィールド幅を割り当てたことになります。ですから、実質使用フィールド幅は85バイトになり、残り（256—85）の171バイトは使われないことになります。

フィールド幅の合計が256バイトになるまで、どのように割り当ててもかまいません。そして割り当てを行う変数は必ず文字変数を使います（扱うデータが数値であっても、です）。

FIELD文によって、バッファ内に変数のエリアが決まりましたが、書き込む前にもう1つ準備が必要です。

それは、このFIELD文の定義に従って、書き込むデータをバッファ上にセットしなければなりません。つまり、FIELD文で使われている変数に、それぞれ対応するデータをはめこむのです。

LSET
RSET

この、データをバッファ上にセットする方法に次の2つがあります。

```
❶ LSET A1$=X$  ←── バッファ上のA1$にX$の中味をセット
❷ LSET A2$=Y$  ←── バッファ上のA2$にY$の中味をセット

❸ RSET A3$=Z$  ←── バッファ上のA3$にZ$の中味をセット
```

❶も❷もともにバッファ上の変数に、X＄、Y＄、Z＄に入っている文字列をセットすることに変わりはないのですが、❶の場合はLEFT、つまり左づめにセットされ、❷の場合はRIGHT、つまり右づめにセットされる点が違います。いずれの場合も、セットしようとした文字列が、フィールド幅より長いときは、余分な文字は無視されます。

次に、フィールドは文字変数で定義されていますから、数値データをセットするときには、これを文字型に変換しなければなりません。数値が整数か、単精度か、倍精度かによって、次のように変換方法が異なります。

MKI＄
MKS＄
MKD＄

```
LSET B1$=MKI$(X%)   ← 整数データの変換

LSET B2$=MKS$(Y!)   ← 単精度データの変換

LSET B3$=MKD$(Z#)   ← 倍精度データの変換
```

この場合LSETになっていますが、RSETでもかまいません。

ここで1つ大切な注意があります。整数データ、単精度データ、倍精度データに対応するバッファ上の文字変数（上の例ではB1＄、B2＄、B3＄）はフィールド幅が決まっていて、それぞれ2バイト、4バイト、8バイトとなっています。

ですから、数値データを書き込むときは、数字の型によって次のようにFIELDを定義しておかなければなりません。

数値データのFIELD設定

```
FIELD #1,2 AS B1$,4 AS B2$,8 AS B3$
         ↑         ↑         ↑
         │         │         └── 倍精度型の場合
         │         └──────────── 単精度型の場合
         └────────────────────── 整数型の場合
```

これでデータ書き込みの下準備ができました。

それではさっそく実践してみましょう。

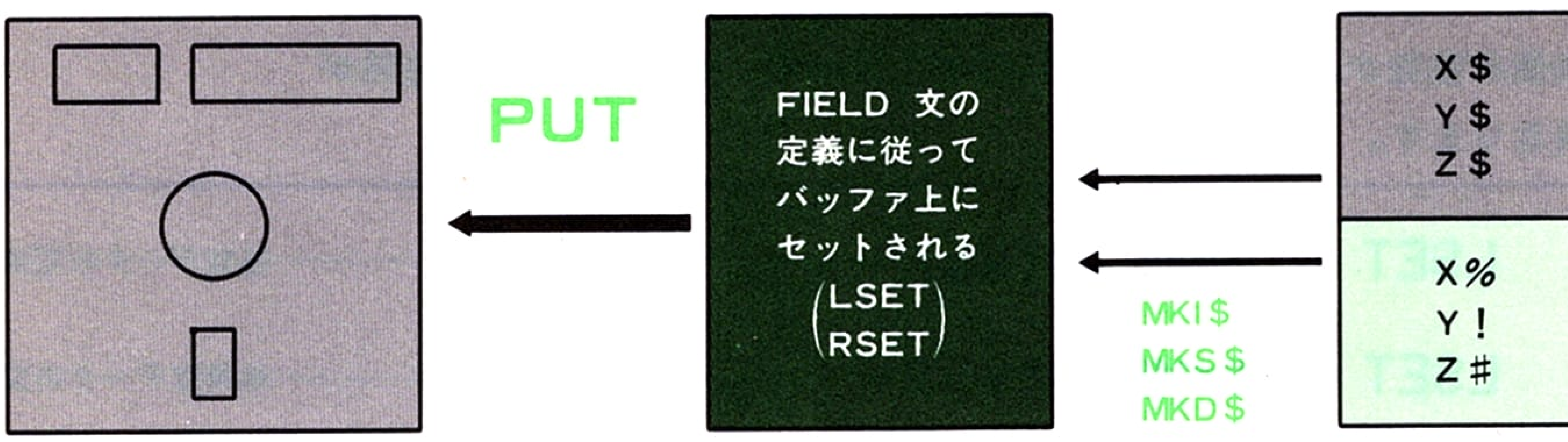
データを書き込んでみよう

シーケンシャルファイルではWRITE＃ 文を使って書き込みをしましたが、ランダムファイルではPUT文を使います。書式は次のとおりです。

```
PUT #1,S
    ↑  ↑
    │  └── レコード番号
    └───── ファイル番号
```

ファイル番号で指定されたバッファの内容（256バイト分）をSで示されるセクタ番号のところへ書き込む（PUTする）ということになります。レコード番号は省略することもできますが、その場合、直前に書き込みが行われたレコードの次にPUTされます。

これまでのプロセスをまとめると次図のようになります。



MKI$、MKS$、MKD$によって文字列に変換された数値データ、および一般の文字変数が、FIELD文で決められたフィールド幅にそれぞれバッファ上でセットされ（LSET、RSET）、このセットされたバッファの内容（256バイト、1セクタ分）がPUT文によってディスクに書き込まれます。

次に示すのは書き込みのテストプログラムです。

```
100 '  test program
110 '
120 OPEN "2:test.dat" AS #1              ←ファイルを開く
130 FIELD #1,2 AS CD$,20 AS NM$,15 AS TEL$  ←フィールドの設定
140 '
150 INPUT "ｺｰﾄﾞ No.(END=9999)";CD      ┐
160 IF CD=9999 THEN GOTO 270           │データ入力
170 INPUT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ";A$              │
180 INPUT "TEL No. ｦ ﾄﾞｳｿﾞ";B$          ┘
190 S=S+1        ←1レコードのデータ入力ごとに、書き込むセレクタ番号を1ずらす
200 '
210 LSET CD$=MKI$(CD)   ┐
220 LSET NM$=A$         │バッファ上にデータをセット
230 LSET TEL$=B$        ┘
240 PUT #1,S     ←ディスクにデータを書き込む
250 '
260 GOTO 150     ←ループ
270 CLOSE #1     ←ファイルを閉じる
280 END
```

プログラムは複雑ではありませんから、ごらんいただければわかると思いますが、書き込みまでの流れは次のようになっています。

データ書き込みの流れ

❶ ファイルを開く（OPENする）
❷ フィールドを設定する（FIEDL文）
❸ データを入力する
❹ バッファ上にデータをセットする
❺ ディスクにデータを書き込む

これがデータ書き込みの一般形です。このパターンさえ覚えてしまえば、あとはディスクをどう有効利用するか、ファイルの形をどうするかなどの課題があるだけで、それほどむずかしくはありません。

コードは4桁の数値で入力することにしてありますので、210行でバッファ上にセットする場合、MKI＄を使っています。

書き込むセクタ番号は、コード、名前、電話番号一式を入力するごとに1ずつカウントされ、ずれていきます。

このプログラムでは、コードで2バイト、名前で20バイト、電話番号で15バイトとっていて、これを1レコードとしていますから、全部で37バイト、したがって256から37を引いた残り219バイトは未使用のままです。

それではプログラムをRUNさせて、次の6名のデータを入力してみましょう。再び、前章の方々に登場してもらいました。

```
0001    トクカ゛ワ イエヤス    000-000-0000
0101    トヨトミ ヒテ゛ヨシ    03-222-0000
1002    オタ゛ ノフ゛ナカ゛    045-111-3333
3012    アケチ ミツヒテ゛      2222-55-4444
0133    アライ ハクセキ        03-000-5555
9012    カイハ゛ラ エッケン    444-66-1111
```

```
ｺｰﾄﾞ No.(END=9999)? 0001
ﾅﾏｴ ｦ ﾄﾞｳｿﾞ? ﾄｸｶﾞﾜ ｲｴﾔｽ
TEL No. ｦ ﾄﾞｳｿﾞ? 000-000-0000
ｺｰﾄﾞ No.(END=9999)? 0101
ﾅﾏｴ ｦ ﾄﾞｳｿﾞ? ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ
TEL No. ｦ ﾄﾞｳｿﾞ? 03-222-0000
ｺｰﾄﾞ No.(END=9999)? 1002
ﾅﾏｴ ｦ ﾄﾞｳｿﾞ? ■
```

上の画面は入力中の様子を示したものです。

コード、氏名、電話番号の1ブロックの入力が終るたびにカチッと音がしてディスクへの書き込みが行われます。6つのデータですから、最後のENDマーク（9999）が入力されるまでに、6レコード書き込まれたことになります。

入力が終りましたら、次の画面のステートメントをダイレクトで入力してみてください。いずれも、キーイン後リターンキーを押してください。

```
open "2:test.dat" as #1    ← いま書き込んだファイルをOPEN
Ok
print lof(1)               ← ファイルの長さを調べる
 6                         ← セクタ数
Ok
close                      ← 必ず閉じることを忘れないでください
Ok
■
```

LOF関数

新しい関数がでてきました。LOF関数です。

これは、ファイルの長さを知るための関数で、PRINT文で上のように入力すると、すぐ下に長さを表示してくれます。上の画面では6となっていますね。

ちょっとわかりかけてきたのではありませんか？

そうです、私たちがいま入力したのは6名分のデータ、レコードにして6レコードだったのです。そしてこの場合、1レコードで256バイト、つまり1セクタを使っていましたから、データファイル「2：test.dat」分として全部で6セクタ、ディスク上を占有していたのです。

ここでもう1つ調べてみたいことがあります。

ドライブ2に入っているデータディスクのファイル名を画面に表示してみてください。FILESコマンドでよかったですね。

```
files 2
dskut2.n88 9     EITAN1 dat 1
TELNO1 dat 1     test   dat 1
Ok
```

クラスタ数

いま書き込んだデータファイル

私たちが、いま書き込んだデータファイルがありますね。そのうしろの数字を見てください。1となっています。

クラスタ

この数字の単位はクラスタといいます。

33ページをごらんください。

8インチのフロッピィディスクでは1クラスタが1周分で26セクタ、5インチ両面倍密度のディスクおよび5インチ片面倍密度のディスクでは、半周分の8セクタが1クラスタになっています。

ファイルはすべて、このクラスタ単位でディスク上に確保されますから、たとえば5インチ両面倍密版の場合、1セクタだけ使っても、8セクタ使っても、やはり1クラスタだけそのファイルのエリアとして占有されてしまうのです。あるファイルをOPENして、1文字だけ書き込んでも、やはり1クラスタとられてしまいます。

ファイルがクラスタ単位でディスク上にそのエリアを確保することはわかっていただけましたか?

このことをさらに明確にするために、1つ実験をしてみましょう。

ここでは、5インチ両面倍密度のディスクの場合を例にとります。

先ほどのテストプログラムに、次のように245行を追加してください。カウンタを6にしておきます。

```
145 S=6
```

これで入力の準備ができました。

次に、3名のデータを入力します。3人のキーインが終ったら「9999」を入力して終了させてください。

❶　1111　　ｶｲﾎ　ｾｲﾘｮｳ　　555-111-1111
❷　5000　　ﾖｼﾀﾞ　ｹﾝｺｳ　　333-333-3333
❸　7001　　ｻﾄｳ　ｲｯｻｲ　　222-222-2222

カウンタSを6に初期設定して、7、8、9、というようにセクタ番号をふやしているわけですが、これはどういうことかといいますと、すでに6セクタ分書き込まれているわけですから、さらに7、8、9セクタと書き込んでいきますと、8と9の間でクラスタ数が変わるはずです（1クラスタはミニフロッピィディスクの場合8セクタでしたね）。

一方、ファイルはクラスタ単位で確保されると書きました。ですから、9セクタ分書き込めば、データファイル「2：test.dat」は2クラスタを確保することになります。

これさえ確かめられれば、いままでの説明をご理解いただけると思います。

さあ、いま説明した手順でデータを書き込んでみてください。

いいですか？

入力が終りましたら、再びFILESコマンドでデータディスクのファイル名を画面に表示してみてください。

```
files 2
dskut2.n88 9     EITAN1 dat 1
TELNO1  dat 1    test   dat 2
Ok
█
```

クラスタ数がふえている

どうですか？

これではっきりしましたね。

ランダムファイルのデータを読む

さて、私たちは9名分のデータをランダムファイルに書き込むのに成功しました。でも、ほんとうに書き込まれているかどうか、ちょっと心配ですね。読み出して画面に表示してみましょう。

ランダムデータを読み出すには、GET文を使います。GETするのです。なんとなくわかるような感じがするでしょう？

そして書き込みのときにレコード（セクタ）単位でPUTしたように、読み込むときもレコード単位でGETすればよいのです。

書式は次のとおりです。

GET文

```
GET #1,R
```

R：レコード番号(ファイル内でのセクタ番号)
#1：ファイル番号

この書式の意味は、ファイル番号で指定されたランダムファイルのRセクタ目のデータを、ファイル番号に対応するバッファ上に読み出す、ということです。

もう少し具体的にいいますと、上の例は現在OPENしている＃1のランダムファイルの第Rセクタを＃1のバッファ上に取り出すということになります。

ファイル番号は1としてありますが、この数は0から15までの数を指定できます。ただし、How many files（0－15）？で入力した数を超えてはいけません。

突然GET文のことを書きましたので、中には、何もしなくてもすぐGETできるのか、と誤解する方もいるかもしれませんので補足しておきますが、データを読むときも書き込みのときと同じように、最初にファイルをOPENして次にフィールドを設定しておかなくてはなりません。

そして、このFIELD文によるフィールドの定義がなされていると、あるセクタをGETしたときに、自動的に、それぞれ定義されたフィールド幅をもつバッファ上の変数に割り当てられるのです。書き込みのときにはLSETまたはRSETを使ってセットしなければなりませんでしたが、読むときには必要ないのです。

このようにバッファ上に取り出され、バッファ上の変数に割り当てられたランダムデータを画面に表示するには、文字型データはそのまま使えますが、数値データは書き込みの際に文字型に変換されている

＊VAL関数も数値変換に使われますが、MKI$などで文字列に変換してあるものは必ずCVI、CVS、CVDを用いて数値に戻さなければなりません。

わけですから、もう一度数値に戻してやらなければなりません。

たとえば、B1＄、B2＄、B3＄というバッファ上の変数に、それぞれ2バイト、4バイト、8バイトの数値が文字型として代入されているとしますと、CVI、CVS、CVD＊を用いて次のように数値に戻してやらなければなりません。

```
X%=CVI(B1$) ←――――――――― 整数値に変換

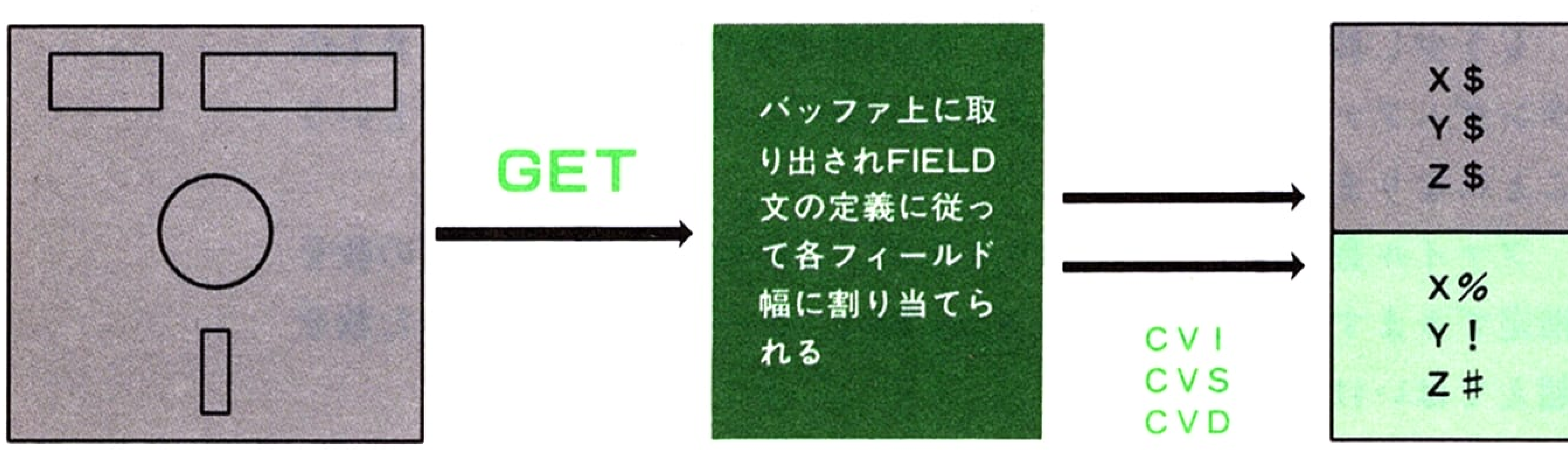
Y!=CVS(B2$) ←――――――――― 単精度型数値に変換

Z#=CVD(B3$) ←――――――――― 倍精度型数値に変換
```

こうして数値型に変換しておきますと、N88-BASICモードで使っていたのと同じ扱いができるのです。

そして最後に、ファイルを使い終ったらCLOSEです。

データを読む過程を図にすると次のようになります。



それではデータ読み出しを実行する前に、その手順を箇条書きにして整理しておきましょう。

❶ ランダムファイルを開く(OPENする)。

❷ FIELD文でフィールドを定義する。

❸ GET文を用いて1レコード(256バイト)ずつバッファ上に取りだす。

❹ 数値データの場合にはCVI、CVS、CVDを使って文字型を数値型に変換する。

❺ ランダムファイルを閉じる(CLOSE)する。

さあ準備完了です。先ほど書き込んだデータを読み出して画面に表示してみましょう。

プログラムを次に示します。行番号を300からとして、データ書き込み部分のうしろにつけ加えましょう。

```
300 ' random data read
310 OPEN "2:test.dat" AS #1
320 FIELD #1,2 AS CD$,20 AS NM$,15 AS TEL$
330 FOR I=1 TO LOF(1)
340 GET #1,I
350 PRINT USING "####  ";CVI(CD$);
360 PRINT NM$;
370 PRINT TEL$
380 NEXT I
390 CLOSE #1
400 END
```

ファイルをオープン（310行）

フィールドの設定（320行）

1セクタから最終セクタまでループ（330行）

画面に表示（350〜380行）

ファイルを閉じる（390行）

330行のLOF（1）は、1番のファイルの長さ、つまり何セクタ使っているか、その値が与えられます。そして、その値の示す回数だけ340行のGET文によってディスク上のデータがバッファ上に取り出されます。これをPRINT文で画面に表示すればよいわけですが、CD＄だけは数値に変換しなければいけませんから、350行でCVIを使って変換しています。

これを実行するときはRUNコマンドではなく*、次のようにしてください。RUNコマンドですと、このステートメントの前に書き込み用のステートメントがありますから、そちらが実行されてしまいます。

＊もしRUNコマンドで実行してしまったときはストップキーを押してCLOSEをキーインし、再びリターンキーを押しておいてください

```
go to 300 ⏎
```

次のような画面になりましたか？

```
go to 300
   1  トクガ゛ワ イエヤス         000-000-0000
 101  トヨトミ ヒデ゛ヨシ         03-222-0000
1002  オダ゛ ノブ゛ナガ゛         045-111-3333
3012  アケチ ミツヒデ゛           2222-55-4444
 133  アライ ハクセキ             03-000-5555
9012  カイバ゛ラ エッケン         444-66-1111
1111  カイホ セイリョウ           555-111-1111
5000  ヨシダ゛ ケンコウ           333-333-3333
7001  サトウ イッサイ             222-222-2222
Ok
■
```

ちゃんと書き込みができていたようですね。

しかし、コード番号の並びが少々ふぞろいですね。

350行でPRINT USING文を使っていますから桁ぞろいはできているのですが、前がデコボコです。これをきれいに整とんするには、次の345行、346行をキーインし、350行を少し変更すればうまくいきます。

やってみましょう。

数値を文字列に変換

```
345 CD=CVI(CD$):CO$=MID$(STR$(CD),2)
346 CO$=RIGHT$("000"+CO$,4)
350 PRINT CO$"  ";
```

4桁の文字列にする

STR$
MID$
RIGHT$

STR$関数は数値を文字列に変換しますが、数値の前にかくされている符号も含めて変換しますので、つまり1番目はプラスの符号になっていますから、2番目以降を取り出さなければなりません。そしてこれに3桁の文字例「０００」をつなげ、右側から4桁抜き出せば4桁の文字列としてのコードができあがるわけです。

実行して確認してみてください。

ここで1つ実験をします。330行を次のように変えて実行してみてください。

```
330 FOR I=1 TO LOF(1)+1
```

```
7001   ｻﾄｳ ｲｯｻｲ              222-222-2222
Input past end in 340
Ok
close
Ok
█
```

エラーメッセージ

〈注〉この節で扱ったランダムファイルへの書き込み、および読み出しのプログラムはテスト用ですから消してしまってかまいません。

エラーメッセージがでてストップしてしまいました。

まず、CLOSEしておいてください。

これは、340行で、「ファイルの最終レコードを超えて読み出そうとしている」という意味のエラーです。ファイルを扱うようになると、このエラーはよく出会うエラーです。ここではファイルの長さLOF(1)にプラス1しているのですから、エラーの原因が1目でわかりますが、ここがVとかYとかといった変数だと、しばし考え込むことがよくあります。

4・2　メニュープログラムとデータの書き込み

この節では、前節で得た知識をもう一度確認する意味で、簡単な宛て名書き印刷をやってみます。

データファイルには、次の項目を保存しておきます。

❶ **コード番号**

❷ **氏名**

❸ **郵便番号**

❹ **住所**

❺ **番地**

これらを1レコードとして1セクタ内に書き込みます。少しムダな使い方ですが、これは学習のためですし、慣れるためにはある程度簡素化しておいたほうがよいと思うからです。

宛て名印刷の方法は、宛て名印刷をしたい人のコードを次々と入力し、エンドマークを入れると、あとはプリンタがかってに印刷をしてくれるというふうにします。

コード番号には、いろいろなつけ方があります。下2桁を県別にするとか、職種の区別に第何桁を使うとか、また使用桁数もケースによってだいぶ違いがでてくると思います。しかし、この本では、データの入出力をどう扱うか、という点にテーマをしぼっていますので、コードのつけ方にはとらわれずに話を進めることにいたします。

メニュープログラム

メニュー画面は「menu01.bas」を少し手直しして、次のようにしました。

```
アテナ インサツ プログラム

データ ノ ニュウリョク・・・・・・・1
データ ノ シュウセイ・・・・・・・・2
アテナ ノ インサツ・・・・・・・・・3
オワリ・・・・・・・・・・・・・・・・4

input job number ? ■
```

メニュープログラムのリストを次に示します。手直しした個所にはコメントを入れておきました。

```
10 ' file name ==> menu03.bas
20 ' 1984.1.25
30 ' Menu program to print address
40 ' save "menu03.bas"
50 '
100 CONSOLE 0,25,0,0:WIDTH 40,25
110 DEFINT A-Z
120 CLS
130 '
140 READ MENU
150 DIM M$(MENU)
160 FOR I=1 TO MENU
170 READ M$(I)
180 NEXT I
190 '
200 LOCATE 7,4:PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,5:PRINT "│  ｱﾃﾅ ｲﾝｻﾂ ﾌﾟﾛｸﾞﾗﾑ  │"
230 LOCATE 7,6:PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270 LOCATE 8,7+I:PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13:INPUT "input job number ";NO
310 LOCATE 8,15:PRINT M$(NO)
320 LOCATE 18,16:INPUT "OK (y/n)";Y$
330 IF Y$="n" OR Y$="N" THEN CLS:GOTO 200
340 ON NO GOTO 360,370,380,390
350 '
360 RUN "DMAIL1.bas"
370 RUN "DMAIL2.bas"
380 RUN "DMAIL3.bas"
390 CLS
400 LOCATE 15,7:PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
410 END
```

ファイル名変更

見出しの変更

プログラム名変更

```
1000 '
1010 '/// data area ///
1020 '
1030 DATA 4
1040 DATA データ ノ ニュウリョク・・・・・・・1
1050 DATA データ ノ シュウセイ・・・・・・・・2
1060 DATA アテナ ノ インサツ・・・・・・・・・・3
1070 DATA オワリ・・・・・・・・・・・・・・・・・・4
```

メニューの変更

データ入力部分のプログラム

プログラムの先頭部分は、いつものようにファイル名、作成年月日を変えたりするだけでよいでしょう。

```
10 ' file name ==> DMAIL1.bas
20 ' 1984.1.25
30 ' random data write
40 ' save "DMAIL1.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
120 CLS
```

次にデータ書き込み用のランダムファイルをOPENしなければなりませんが、各項目が使用するフィールドを次のように設定することにします。

●フィールドの割り当て●

氏名		郵便番号		住所		番地	
変数	バイト	変数	バイト	変数	バイト	変数	バイト
NM$	20	PN$	6	AD$	35	BA$	20

ここで、おやッと気づいたことはありませんか？

そうです、129ページのファイルに書き込む項目にはコードも入っていましたが、この表には見当たりませんね。コードは書き込まないでいいのでしょうか？　この理由はすぐ説明します。

とにかくフィールドが定義できましたから、ランダムファイルをOPENする部分をコーディングしてみましょう。

```
130 '
140 *OPNF
150 OPEN "2:DMAIL1.dat" AS #1
160  FIELD #1,20 AS NM$,6 AS PN$,
           35 AS AD$,20 AS BA$
```

インデックス

ランダムファイルにおいては、インデックスという考え方をよく使います。技術関係の本には索引がついていますね。あの索引と同じと考えていただいてよいと思います。

ある専門用語を本文中から探そうとしても、ページをめくっただけではなかなか見つけることができません。ところが索引を調べれば、その用語にページ数が書いてありますから、すぐに探すことができますね。ランダムファイルで使うインデックスも同じことなのです。

たとえば、次の4名のデータを1人を1レコード（1セクタ）としてランダムファイルに書き込むとします。

コード	氏名	郵便番号	住所	番地
0011	ﾄｸｶﾞﾜ ｲｴﾔｽ	101	aaaaaaaaaa	1-2-3
1122	ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ	030	bbbbbbbbbbbb	456-78
3344	ｵﾀﾞ ﾉﾌﾞﾅｶﾞ	231	cccccccccc	1984ﾊﾞﾝﾁ
5566	ｱｹﾁ ﾐﾂﾋﾃﾞ	680	dddddddddd	9-8-76

このとき、ランダムファイルの考え方では、「コード」「氏名」「郵便番号」「番地」、この5項目を1セクタ内にすべて書き込むのではなく、コードだけはシーケンシャルファイルに書くことにしているのです。

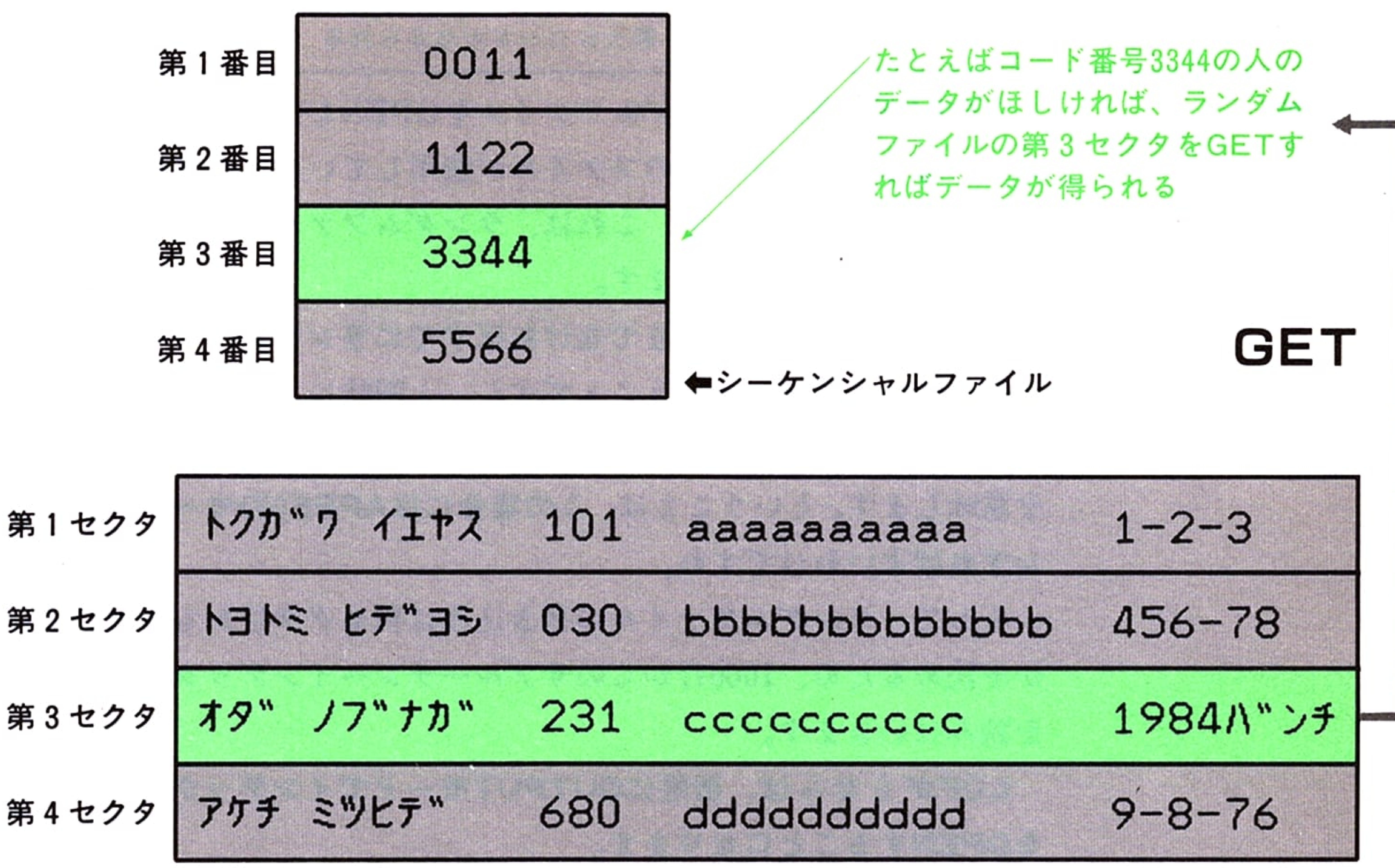
そうしますと、このシーケンシャルファイルに書き込まれたコードが、本でいう索引の役割をするのです。

なぜかといいますと、シーケンシャルファイルは連続的に前のほうから順序よくデータがつめこまれていました。覚えていますか？

つまり、上の例でいいますと、「0011」は1番目に、「1122」は2番目に、「3344」は3番目に、「5566」は4番目に書き込まれます。と同時に、そのコードに対応する「トクガワ」「トヨトミ」「オダ」「アケチ」は、ランダムファイルの第1セクタ、第2セクタ、第3セクタ、第4セクタに書き込まれます。ということは、キーボードからコードを入力したとき、そのコードがシーケンシャルファイルのX番目にあったとしますと、ランダムファイルのXセクタ目をGETすれば、希望する

人のデータが得られるわけです。

このことを図解したのが次の図です。



ご理解いただけたでしょうか？

要するに、データをランダムファイルにセーブするときに、インデックス用のシーケンシャルファイルも一緒に作っておくのです。

さて、メニュープログラムで1番の「データの入力」を選択すると、この「DMAIL1.bas」が実行されるわけですから、初回の書き込みも複数回目の書き込みも、いずれも考慮してコーディングしなければなりません。

OUTPUTモード

初回でしたら、インデックス用のシーケンシャルファイルはOUTPUTモードでよいですが、2回目、3回目にOUTPUTモードでオープンすると、それまでに作ったインデックスファイルが壊れてしまいます（OUTPUTは新規にファイルに書くときのモードでした)。

それでは常にAPPENDモードでOPENすればよいのでしょうか。一見正しいように思えますが、そうではないのです。

たとえば、初回の入力時にAPPENDモードでOPENすると、

```
File not found
```

というエラーメッセージが返ってきます。

そこで、初回でも複数回目でも入力可能なようにするには、ちょっ

とした細工が必要なのです。次のステートメントを入れておきます。

```
170  IF LOF(1)<>0 THEN GOSUB *COUNT:GOTO 200
```

LOF(1) ──最大レコードが与えられる

LOF関数

このLOF関数は、ランダムファイルをOPENしたときに、最大のレコード番号（あるいはそのファイルが使用しているセクタ数）を返してくるものです。そして、これは、ランダムファイルを初めてオープンしたときには0を与えます。

ですから、**LOFの値が0でなければすでにランダムファイルの書き込みは行われている**ということですし、と同時に、それはインデックス用のシーケンシャルファイルもすでにディスクの中に存在することを意味します。ということは、**この場合にはAPPENDモードでオープンすればよい**わけですね。

そして、ランダムファイルの書き込みは何セクタ目からすればよいかを決めるため、1000行からのサブルーチンへインデックスファイルを読みにいきます。

LOFが0ならば、新規にOUTPUTモードでインデックスファイルをOPENすることになります。

1000行からのサブルーチンは次のようになります。

```
1000 '
1010 *COUNT
1020 OPEN "2:INDEX1.dat" FOR INPUT AS #2
1030 FOR I=1 TO 2 STEP 0
1040  IF EOF(2) THEN I=3:GOTO 1070
1050   INPUT #2,CO
1060   REC=REC+1
1070 NEXT I
1080 SEC=REC+1
1090 CLOSE #2
1100 RETURN
```

1040行 ← 読み込むデータがなければI＝2にしてFOR～NEXTループから抜ける

1060行 ← データ数をカウントする

1080行 ← 総データ数にプラス1した値が入力データを書き込むランダムファイル上のレコード位置になる

1030行のFOR文はステップを0にしてあります。つまり、**ループ変数Iは常に1のままで2に進めず、1040行の条件が満たされるまで、FOR～NEXTループをくりかえします。**WHILE～WENDによく似たおもしろいテクニックです。WHILE～WENDは使いませんでしたが、ここでは、こんなやり方もあるということを覚えてください。

1080行のSEC＝REC＋1は、前ページの図を見ていただけばわかると思いますが、ランダムファイルへの書き込み位置（PUTする位置）はシーケンシャルファイルのデータ数プラス1になります。

RETURN文によりサブルーチンから帰った場合は、200行からのAPPENDモードへ、そうでない場合は、初回のデータ入力ということになりますから、OUTPUTモードでインデックスファイルをOPENします。

```
180 OPEN "2:INDEX1.dat" FOR OUTPUT AS #2
190 SEC=1:GOTO *DAT.INPUT
200 OPEN "2:INDEX1.dat" FOR APPEND AS #2
```

190行でランダムファイルへの書き込み位置を示す変数SECは１となっていますが、この場合、初回の書き込みですから、ファイルの第１セクタからPUTしていきます。

次はデータ入力と書き込み部分です。

```
210 '
220 *DAT.INPUT
230 LOCATE 10,5:PRINT "ﾃﾞｰﾀ ﾉ ﾆｭｳﾘｮｸ ｦ ｼﾏｽ"
240 FOR I=1 TO 5000:NEXT I
250 CLS
260 PRINT "ｺｰﾄﾞ No. ｦ ﾄﾞｳｿﾞ (END=9999)"
270 INPUT "    ==> ";CD
280  IF CD=9999 THEN GOTO 490
290 PRINT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ"
300 INPUT "    ==> ";A1$
310 PRINT "ﾕｳﾋﾞﾝﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ"
320 INPUT "    ==> ";A2$
330 PRINT "ｼﾞｭｳｼｮ ｦ ﾄﾞｳｿﾞ"
340 INPUT "    ==> ";A3$
350 PRINT "ﾊﾞﾝﾁ ｦ ﾄﾞｳｿﾞ"
360 INPUT "    ==> ";A4$
370 PRINT
380 INPUT "        OK ﾃﾞｽｶ  ( y/n )";ANS$
390 IF ANS$="n" OR ANS$="N" THEN GOTO 250
400 '
410 *DAT.WRITE
420 WRITE #2,CD
430 LSET NM$=A1$ :LSET PN$=A2$
440 LSET AD$=A3$ :LSET BA$=A4$
450 PUT #1,SEC
460 SEC=SEC+1
470 GOTO 250
```

（260～370行）データ入力
（420行）インデックスファイルにコードを書き込む
（430～440行）バッファ上にデータをセット
（450行）ランダムファイルにデータを書き込む
（460行）次の書き込みのためにセクタ数をふやす

データ入力を終りたいときは270行で9999を入力するようにしています。

420行、450行でもわかるように、シーケンシャルファイルは＃2のバッファ、ランダムファイルは＃1のバッファを使っています。1つのバッファを2つ以上のファイルで共用することはできませんから注意してください。

この段階では、ファイルが2つ同時にOPENしていますね。

あなたは、ディスク起動時の

```
How many files(0-15)? █
```

というメッセージに対して、どんな数をキーインしましたか？　その数を超えたバッファを同時に使うことはできません。つまり、オープニング時に入力した数字は、バッファ数と考えることもできるのです。

次は、ファイルを閉じてエンディング、そして総リストです。

```
480 '
490 *F.CLOSE
500 CLOSE #1,#2
510 '
520 CLS
530 LOCATE 8,8:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
540 LOCATE 8,9:INPUT "  ( y/n )";ANS$
550 IF ANS$="y" THEN RUN "menu03.bas"
560 END
```

500行：1番、2番ファイルを閉じる

550行：ここにOR ANS＄＝"Y"を入れておけば大文字でも小文字でも動作します。

```
10 ' file name ==> DMAIL1.bas
20 ' 1984.1.25
30 ' random data write
40 ' save "DMAIL1.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
120 CLS
130 '
140 *OPNF
150 OPEN "2:DMAIL1.dat" AS #1
160  FIELD #1,20 AS NM$,6 AS PN$,
          35 AS AD$,20 AS BA$
170  IF LOF(1)<>0 THEN GOSUB *COUNT:GOTO 200
180 OPEN "2:INDEX1.dat" FOR OUTPUT AS #2
```

```
190 SEC=1:GOTO *DAT.INPUT
200 OPEN "2:INDEX1.dat" FOR APPEND AS #2
210 '
220 *DAT.INPUT
230 LOCATE 10,5:PRINT "ﾃﾞｰﾀ ﾉ ﾆｭｳﾘｮｸ ｦ ｼﾏｽ"
240 FOR I=1 TO 5000:NEXT I
250 CLS
260 PRINT "ｺｰﾄﾞ No. ｦ ﾄﾞｳｿﾞ (END=9999)"
270 INPUT "    ==> ";CD
280  IF CD=9999 THEN GOTO 490
290 PRINT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ"
300 INPUT "    ==> ";A1$
310 PRINT "ﾕｳﾋﾞﾝﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ"
320 INPUT "    ==> ";A2$
330 PRINT "ｼﾞｭｳｼｮ ｦ ﾄﾞｳｿﾞ"
340 INPUT "    ==> ";A3$
350 PRINT "ﾊﾞﾝﾁ ｦ ﾄﾞｳｿﾞ"
360 INPUT "    ==> ";A4$
370 PRINT
380 INPUT "      OK ﾃﾞｽｶ  ( y/n )";ANS$
390 IF ANS$="n" OR ANS$="N" THEN GOTO 250
400 '
410 *DAT.WRITE
420 WRITE #2,CD
430 LSET NM$=A1$ :LSET PN$=A2$
440 LSET AD$=A3$ :LSET BA$=A4$
450 PUT #1,SEC
460 SEC=SEC+1
470 GOTO 250
480 '
490 *F.CLOSE
500 CLOSE #1,#2
510 '
520 CLS
530 LOCATE 8,5:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
540 LOCATE 8,6:INPUT "  ( y/n )";ANS$
550 IF ANS$="y" THEN RUN "menu03.bas"
560 END
1000 '
1010 *COUNT
1020 OPEN "2:INDEX1.dat" FOR INPUT AS #2
1030 FOR I=1 TO 2 STEP 0
```

```
1040  IF EOF(2) THEN I=3:GOTO 1070
1050   INPUT #2,CO
1060   REC=REC+1
1070 NEXT I
1080 SEC=REC+1
1090 CLOSE #2
1100 RETURN
```

プログラムをセーブしたら、下に示すデータを入れて、動作テストをしてみましょう。

また、途中でエラーがでてストップしたら、ダイレクトで一度CLOSEしてから、チェックしてください。バグ（プログラムの間違い）をとってから再度RUNする前に、すでにデータファイルがディスクに作られていたら、それをKILLコマンドで削除してから始めましょう。正常に動くまでは、めんどうでも、こういったくりかえしが必要です。

バグ

KILL

```
kill "2:INDEX1.dat" ⏎

kill "2:DMAIL1.dat" ⏎
```

0011	ﾄｸｶﾞﾜ ｲｴﾔｽ	101	aaaaaaaaaa	1-2-3
1122	ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ	030	bbbbbbbbbbbb	456-78
3344	ｵﾀﾞ ﾉﾌﾞﾅｶﾞ	231	cccccccccc	1984ﾊﾞﾝﾁ
5566	ｱｹﾁ ﾐﾂﾋﾃﾞ	680	dddddddddd	9-8-76

```
ｺｰﾄﾞ No. ｦ ﾄﾞｳｿﾞ (END=9999)
    ==> ? 0011
ﾅﾏｴ ｦ ﾄﾞｳｿﾞ
    ==> ? ﾄｸｶﾞﾜ ｲｴﾔｽ
ﾕｳﾋﾞﾝﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ
    ==> ? 101
ｼﾞｭｳｼｮ ｦ ﾄﾞｳｿﾞ
    ==> ? aaaaaaaaaa
ﾊﾞﾝﾁ ｦ ﾄﾞｳｿﾞ
    ==> ? 1-2-3■
```

4・3 データの修正

住所の変更とか、入力ミスにあとで気づいたとかした場合、プログラムさえしっかり作っておけば、データの修正は簡単です。

シーケンシャルファイルのときは、氏名の一部を入力して、それと同じ文字列をファイルの中から探して修正を加えていました。カタカナでの入力はちょっとめんどうですし、すでに書き込みずみのデータをすべて配列の中にとり込んでおかなければいけませんでした。

ところがランダムファイルとなると、キー入力はきわめて簡単。数字列のコードをテンキーで入力すればよいのですし、コードは氏名と1対1の対応をなしていますから、同姓の人を考慮する必要がありません。ただ、コード体系を完成させるまでは少々大変だと思いますが、あとで楽をするためには、初めに苦労はつきものですね。

それに、ディスクに保存できるデータ量もシーケンシャルファイルと比べて格段の差がでてきます。シーケンシャルファイルではすべてのデータを配列に読み込まなければならないので、コンピュータのメモリを考慮すれば、そこにおのずと制限がでてきます。

しかし、ランダムファイルになりますと、シーケンシャルに書き込まれているインデックスは配列にとり込む必要がありますが、その他のデータの入出力は、すべて256バイトのバッファだけで事がすんでしまいます。大変な違いですね。

ちなみに一番容量の少ない片面倍密度のミニフロッピィディスクの場合、システムを入れずにデータファイルディスクとしてだけ使うのでしたら、34トラック*使えますから、容量は68クラスタということになります（1トラック＝2クラスタでしたね）。

これをセクタ数に換算すると、1クラスタ＝8セクタですから、68を8倍して544セクタとなります。

この章では1セクタに1名のデータしか書き込んでいませんが、もし3名分書き込むとすれば、

544（セクタ）×3（名）＝1632（名）

の資料をストックできるのです。

一番容量の少ないディスクでさえ、このデータ量ですから、シーケンシャル的な考え方ではとても手におえません。

実用面でデータを管理するには、やはりランダム的な思考法が要求されるのです。

＊実際には35トラックありますが、このうち1トラックはファイル管理のためのデータが書き込まれますので使うことができません。

プログラムの作成

この部分はもう、例によって例のごとくといった感じですね。

```
10 ' file name ==> DMAIL2.bas
20 ' 1984.1.25
30 ' random data correct
40 ' save "DMAIL2.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
120 CLS
130 '
140 OPEN "2:DMAIL1.dat" AS #1
150  FIELD #1,20 AS NM$,6 AS PN$,
           35 AS AD$,20 AS BA$
```

ファイル名

ランダムファイルOPEN

フィールドの設定

データを修正する場合には、目的のデータを探し出さなければなりませんから、インデックス用の配列を作って読み込む必要があります。それが次のステートメントです。

```
160 '
170 GOSUB *R.INDEX:FLAG=1
180 DIM A(DM):DM=0:GOSUB *R.INDEX:INDEX=DM

1000 '
1010 *R.INDEX
1020 '
1030 OPEN "2:INDEX1.dat" FOR INPUT AS #2
1040 FOR I=1 TO 2 STEP 0
1050  IF EOF(2) THEN I=3:GOTO 1090 ELSE DM=DM+1
1060  IF FLAG=1 THEN GOTO 1080
1070  INPUT #2,CO:GOTO 1090
1080  INPUT #2,A(DM)
1090 NEXT I
1100 CLOSE #2
1110 RETURN
```

170行のGOSUB文で1000行からのサブルーチンへ飛び、インデックスファイルに書き込まれているインデックス(コード番号)の数を読みます。これは、インデックス用の配列の大きさを決めるためです。

配列はあらかじめプログラムの先頭部分で、A(100)、A(500)のよ

うに宣言しておいてもよいのですが、メモリ効率からいくと、170行、180行でのやり方のほうがよいでしょう。

170行のGOSUB文により1000行からのサブルーチンへ飛んだ段階では、まだFLAG＝0でフラグは立っていませんから、1070行のINPUT＃文を実行し、データ数を読みます。そして、RETURN文で帰ったときにフラグを立て(FLAG＝1)、配列を宣言し、再度同じサブルーチンへ飛び、今度は、いま宣言した配列の中にインデックスを読み込みます。

2度目にサブルーチンへジャンプする前に、DM＝0にクリアしておくのを忘れないでください。こうしておきませんと、1050行でさらに加算が行われますから、1080行のINPUT＃文で、

●プログラミングではフラグという考え方をよく使います。ある条件を満たしたときとか、ある動作を行ったときフラグを立てておけば(1にする)、プログラムの中での条件判断が大変楽です。たとえば、ある家を探すのに目じるしとしてフラグ(旗)を立てておけば探しやすいのと同じです。

```
Subscript out of range
```

のエラーになります。つまり、配列がたりなくなってしまった、というわけです。

これで下準備ができましたから、次は修正したいデータの入力部分になります。

```
190 '
200 LOCATE 5,5:FLAG=0
210 PRINT "ﾄﾞﾅﾀ ﾉ ｼｭｳｾｲ ﾃﾞｽｶ (END=9999)?"
220 LOCATE 7,6:INPUT "code number";CD
230 IF CD=9999 THEN GOTO 300
240 LOCATE 10,8:INPUT "OK ﾃﾞｽｶ ( y/n )";ANS$
250 IF ANS$="n" OR ANS$="N" THEN CLS:GOTO 200
270 GOSUB *DAT.CORRECT
280 CLS:GOTO 200
```

コードが入力されると、240行で確認をうながし、間違いなければ270行で、2000行からの修正サブルーチンへ飛びます。

修正作業を終りたいときは、「9999」を入力するようにします。

サブルーチンのプログラムは次のとおりです。

```
2000 '
2010 *DAT.CORRECT
2020 '
2030 CLS
2040 FOR I=1 TO INDEX
2050  IF A(I)=CD THEN FLAG=1:SEC=I:I=INDEX+1
2060 NEXT I
2070 IF FLAG=0 THEN GOTO 2250
```

(2040〜2060行) 入力されたコードをインデックスの中から検索する

(2070行) コードが見つからないときの処理

```
2080 GET #1,SEC
2090 LOCATE 2,3:PRINT "ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ"
2100 LOCATE 4,4:PRINT "ｼｭｳｾｲ ﾅｹﾚﾊﾞ RET ｷｰ"
2110 PRINT
2120 PRINT NM$:INPUT "   ==> ";A1$
2130  IF A1$="" THEN A1$=NM$
2140 PRINT PN$:INPUT "   ==> ";A2$
2150  IF A2$="" THEN A2$=PN$
2160 PRINT AD$:INPUT "   ==> ";A3$
2170  IF A3$="" THEN A3$=AD$
2180 PRINT BA$:INPUT "   ==> ";A4$
2190  IF A4$="" THEN A4$=BA$
2200 LSET NM$=A1$:LSET PN$=A2$
2210 LSET AD$=A3$:LSET BA$=A4$
2220 PUT #1,SEC
2230 RETURN
2240 '
2250 BEEP:BEEP:BEEP
2260 LOCATE 8,5:PRINT "ｿﾉ ｶﾀ ﾊ ﾄｳﾛｸ ｻﾚﾃｲﾏｾﾝ"
2270 FOR I=1 TO 5000:NEXT I
2280 RETURN
```

2080：該当セクタをGETする
2120〜2190：正しいデータ入力
2200〜2210：バッファ上にセット
2220：ランダムファイルに書き込む
2250：ブザー音

●あるデータが配列内にあるかどうか探し出したいときは SEARCH 関数を使うと便利ですが、この方法はあとで実践することにしましょう。

2040行と2060行のFOR〜NEXT文で、入力されたコードCDと同じものがあるかどうかインデックスを検索します。もし見つかれば、フラグ（FLAG）を立てておき、その見つかったときのIの値が、修正すべきレコードが書き込まれているセクタ数になるわけですから、変数SECに代入しておきます。コードとデータは1対1対応ですから、1つ見つかればそれ以上の検索は不要となり、ループ変数Iの値に最終値INDEXを代入し、FOR〜NEXTループから抜け出してしまいます。

2070行でフラグ(FLAG)が立っていなければ、該当データがなかったわけですから、2250行でブザー音を鳴らし、画面にメッセージを表示します。

2080行では、該当セクタをGETし、修正の準備をします。

データの修正方法は、現在書き込まれているデータを画面に表示し、そのままでよければリターンキーを押し、直したければキー入力するようにしました。

2200行以降で、正しいデータをバッファ上にセットし、ランダムデータファイルに書き込みます。

最後は、使用ファイルのCLOSEです。

```
290 '
300 CLOSE #1
310 '
320 CLS
330 LOCATE 8,5:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
340 LOCATE 8,6:INPUT "    ( y/n )";ANS$
350 IF ANS$="y" THEN RUN "menu03.bas"
360 END
```

OR ANS$＝"Y"を入れれば大文字でも小文字でも動作します

ここでちょっと横道にそれた話を1つ。

これまで、画面に表示されたメッセージに対しての答えは、ほとんどの場合、340行のように「y」または「n」でキー入力させていました。そしてキー入力後リターンキーを押さなければいけませんでしたね。

次の2つのプログラムを比べてください。(注)

〈注〉このプログラムはキーインしなくていいのですよ。参考のためにかかげました。

```
500 INPUT "Do you want any job? ( y/n )";ANS$
510 IF ANS$="y" THEN RUN "menu"
520 PRINT "Have a nice day!"
530 END
```

```
500 PRINT "Do you want any job? ( y/n )"
510 Y$=INKEY$:IF Y$="" THEN GOTO 510
520 IF Y$="Y" OR Y$="y" THEN RUN "menu"
530 IF Y$="N" OR Y$="n" THEN GOTO 550
540 BEEP:BEEP:GOTO 510
550 PRINT "Have a nice day!"
560 END
```

1番目のプログラムは、yを入力しリターンキーを押したときにのみ「menu」が実行されます。2番目はリターンキーを押す必要もないうえ、無関係のキー入力があったときはブザー音を出して、もう一度正しい入力をうながします。お好きなほうをどうぞ。

プログラムの実行とリスト

```
10 ' file name ==> DMAIL2.bas
20 ' 1984.1.25
30 ' random data correct
40 ' save "DMAIL2.bas"
```

```
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
120 CLS
130 '
140 OPEN "2:DMAIL1.dat" AS #1
150  FIELD #1,20 AS NM$,6 AS PN$,
              35 AS AD$,20 AS BA$
160 '
170 GOSUB *R.INDEX:FLAG=1
180 DIM A(DM):DM=0:GOSUB *R.INDEX:INDEX=DM
190 '
200 LOCATE 5,5:FLAG=0
210 PRINT "ﾄﾞﾅﾀ ﾉ ｼｭｳｾｲ ﾃﾞｽｶ (END=9999)?"
220 LOCATE 7,6:INPUT "code number";CD
230 IF CD=9999 THEN GOTO 300
240 LOCATE 10,8:INPUT "OK ﾃﾞｽｶ ( y/n )";ANS$
250 IF ANS$="n" OR ANS$="N" THEN CLS:GOTO 200
270 GOSUB *DAT.CORRECT
280 CLS:GOTO 200
290 '
300 CLOSE #1
310 '
320 CLS
330 LOCATE 8,5:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
340 LOCATE 8,6:INPUT "    ( y/n )";ANS$
350 IF ANS$="y" THEN RUN "menu03.bas"
360 END
1000 '
1010 *R.INDEX
1020 '
1030 OPEN "2:INDEX1.dat" FOR INPUT AS #2
1040 FOR I=1 TO 2 STEP 0
1050  IF EOF(2) THEN I=3:GOTO 1090 ELSE DM=DM+1
1060  IF FLAG=1 THEN GOTO 1080
1070  INPUT #2,CO:GOTO 1090
1080  INPUT #2,A(DM)
1090 NEXT I
1100 CLOSE #2
1110 RETURN
2000 '
2010 *DAT.CORRECT
```

```
2020 '
2030 CLS
2040 FOR I=1 TO INDEX
2050  IF A(I)=CD THEN FLAG=1:SEC=I:I=INDEX+1
2060 NEXT I
2070 IF FLAG=0 THEN GOTO 2250
2080 GET #1,SEC
2090 LOCATE 2,3:PRINT "ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ"
2100 LOCATE 4,4:PRINT "ｼｭｳｾｲ ﾅｹﾚﾊﾞ RET ｷｰ"
2110 PRINT
2120 PRINT NM$:INPUT "   ==> ";A1$
2130  IF A1$="" THEN A1$=NM$
2140 PRINT PN$:INPUT "   ==> ";A2$
2150  IF A2$="" THEN A2$=PN$
2160 PRINT AD$:INPUT "   ==> ";A3$
2170  IF A3$="" THEN A3$=AD$
2180 PRINT BA$:INPUT "   ==> ";A4$
2190  IF A4$="" THEN A4$=BA$
2200 LSET NM$=A1$:LSET PN$=A2$
2210 LSET AD$=A3$:LSET BA$=A4$
2220 PUT #1,SEC
2230 RETURN
2240 '
2250 BEEP:BEEP:BEEP
2260 LOCATE 8,5:PRINT "ｿﾉ ｶﾀ ﾊ ﾄｳﾛｸ ｻﾚﾃｲﾏｾﾝ"
2270 FOR I=1 TO 5000:NEXT I
2280 RETURN
```

138ページで入力した「オダ　ノブナガ」を修正してみましょう。

3344というコード入力後の修正画面です。

```
  ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ
    ｼｭｳｾｲ ﾅｹﾚﾊﾞ RET ｷｰ

ｵﾀﾞ ﾉﾌﾞﾅｶﾞ
  ==> ?                 ← 氏名は修正をしないのでリターンキーのみ押す
231
  ==> ? 000             ← 正しい郵便番号
cccccccccc
  ==> ? xxxxxxxxxx      ← 新住所
1984ﾊﾞﾝﾁ
  ==> ? 0-0-0█          ← 番地の入力
```

4・4 宛て名の印刷

さて、いよいよ最後の印刷プログラムです。

ここで扱うプログラムは、ディスクに登録してあるデータすべての宛て名を印刷するのではなく、ＤＭなどの発送の必要のある人だけをプリンタに出力することにします。

考え方は、対象となる人のコード番号をキーボードから入力し、それを一時、配列に退避させておいて、すべてのキー入力がすんだら、一括して印刷するシステムにします。

１件入力しては印刷する、というようにしてもかまわないのですが、大変時間の浪費になります。キー入力は人間がやらなければなりませんが、それさえすませたら、あとはプリンタにまかせましょう。

とはいっても、これは発送数が相当多数にわたる場合で、本節では2、3のデータをプリンタに打ち出してみるだけですから、プリント時間もつかの間ですが、一応発送数がふえることを想定して考えます。

しかし、データが多いか少ないかの相異だけで、基本的な考え方はまったく同じです。

初期設定

プログラムヘッドはファイル名を変えるだけで、前節のそれと変わりません。

```
10 ' file name ==> DMAIL3.bas
20 ' 1984.1.25
30 ' address print out
40 ' save "DMAIL3.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
120 CLS
130 '
140 OPEN "2:DMAIL1.dat" AS #1
150  FIELD #1,20 AS NM$,6 AS PN$,
             35 AS AD$,20 AS BA$
```

ファイル名

次に配列宣言とデータ読み込みです。

```
160 '
170 GOSUB *R.INDEX:FLAG=1
180 DIM A(DM),B(DM):DM=0:GOSUB *R.INDEX:INDEX=DM
```

この部分のステートメントは、その持つ意味は前節と同じですが、ただ、**配列Bを新しく加えています。**

配列Aはシーケンシャルファイルのインデックスをたくわえておくものでしたが、**配列Bは印刷する必要のあるデータのインデックスをしまっておきます。**先ほど、まとめて印刷するといいましたが、つまり、あとで配列Bにしまわれているコード番号を1つ1つ取り出し、それに対応するランダムファイルのセクタをGETし、プリンタに出力します。

次はコードの入力部分です。

コードの入力

```
190 '
200 LOCATE 10,4:PRINT "ｱﾃﾅ ﾉ ｲﾝｻﾂ ｦ ｼﾏｽ"
210 LOCATE 10,5:PRINT "input code number"
220 LOCATE 10,6:PRINT "   (END=9999)"
230 LOCATE 10,8:INPUT "  ==> ";CD
240 IF CD=9999 THEN GOTO 300
250 LOCATE 10,8:PRINT "
260 N=N+1
270 B(N)=CD
280 GOTO 230
```

260〜270行：配列にコードをしまう

250行：スペースを12個出力して、すぐ前に入力したコードを消す

230行でコードを入力するたびに、260行で配列Bの添字に使う変数Nを1ずつふやし、270行でコードをその配列に登録しています。

240行でエンドマークを判断すると、印刷の処理へ移ります。

```
290 '
300 GOSUB *SET.INX
310 '
320 GOSUB *DAT.PRT
330 '
340 CLS
350 LOCATE 8,5:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 8,6:INPUT "   ( y/n )";ANS$
370 IF ANS$="y" THEN RUN "menu03.bas"
380 END
```

300行：インデックス検索サブルーチンへ

320行：印刷サブルーチンへ

370行：OR ANS$="Y"を入れれば大文字でも小文字でも動作します

300行で2000行からのインデックス検索サブルーチンへ飛び、帰ってきて印刷サブルーチンへ飛びます。印刷が終わると、いつものとおり、ほかの作業をするかどうかのメッセージ表示となります。

次はサブルーチンの入力ですが、**インデックスを読み込む1000—1110行のサブルーチンは144ページと同じですので、そちらを見ながらキーインしてください。**

インデックスの検索　2000行からのサブルーチンを見てみましょう。

```
2000 '
2010 *SET.INX
2020 '
2030 FOR I=1 TO N
2040  FOR J=1 TO INDEX
2050   IF B(I)=A(J) THEN B(I)=J:J=INDEX+1
2060  NEXT J
2070 NEXT I
2080 RETURN
```

ループ変数Iの最大値Nは、登録したデータの個数でしたね。260行で計算されています。

2040行から2060行へかけてのループは、先ほど入力したコードと、インデックスファイルに登録されているコードとを比較し、同じものが見つかれば、そのときのJの値を配列Bにしまうわけです。つまりこの時点で、配列Bからそれまでに入っていたコード番号が追い出され、新しい数値（これが追い出されたコード番号に対応するデータのセクタ番号を示す）が入るわけです。

該当するコードが見つかれば、ループ変数Jの値を最大値にして、このループから抜け出します。余計な仕事はやめようというわけです。

この操作をN個のコードについて行えば、GETするセクタがそれぞれわかったことになりますから、あとの印刷は簡単です。

印刷のサブルーチン

```
3000 '
3010 *DAT.PRT
3020 '
3030 CLS
3040 LOCATE 10,5:PRINT "ｲﾝｻﾂ ｼﾏｽ"
3050 LOCATE 10,6:PRINT "ﾌﾟﾘﾝﾀ ｦ ｾｯﾄ ｼﾃｸﾀﾞｻｲ"
3060 LOCATE 10,8:PRINT "Hit any key if OK"
3070 Y$=INKEY$:IF Y$="" THEN GOTO 3070
3080 FOR I=1 TO N
3090  GET #1,B(I)
3100   LPRINT TAB(5);PN$
3110   LPRINT TAB(5);AD$
3120   LPRINT TAB(15);BA$
3130   LPRINT
3140   LPRINT TAB(10);NM$;TAB(30);" ｻﾏ"
3150  FOR J=1 TO 5:LPRINT :NEXT J
3160 NEXT I
```

3090行：バッファ上にデータをとりだす
3100〜3140行：印刷
3150行：前の宛て名との間に5行空白をとる

```
3170 CLOSE #1
3180 RETURN
```

3070行では、どのキーでも押せば印刷処理に移ります。

3150行は、すぐ前に印刷した宛て名との間に空白行を出力するステートメントです。

次に、プログラム総リストを示します。

```
10 ' file name ==> DMAIL3.bas
20 ' 1984.1.25
30 ' address print out
40 ' save "DMAIL3.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
120 CLS
130 '
140 OPEN "2:DMAIL1.dat" AS #1
150  FIELD #1,20 AS NM$,6 AS PN$,
              35 AS AD$,20 AS BA$
160 '
170 GOSUB *R.INDEX:FLAG=1
180 DIM A(DM),B(DM):DM=0:GOSUB *R.INDEX:INDEX=DM
190 '
200 LOCATE 10,4:PRINT "ｱﾃﾅ ﾉ ｲﾝｻﾂ ｦ ｼﾏｽ"
210 LOCATE 10,5:PRINT "input code number"
220 LOCATE 10,6:PRINT "    (END=9999)"
230 LOCATE 10,8:INPUT "   ==> ";CD
240 IF CD=9999 THEN GOTO 300
250 LOCATE 10,8:PRINT "                "
260 N=N+1
270 B(N)=CD
280 GOTO 230
290 '
300 GOSUB *SET.INX
310 '
320 GOSUB *DAT.PRT
330 '
340 CLS
350 LOCATE 8,5:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
360 LOCATE 8,6:INPUT "    ( y/n )";ANS$
370 IF ANS$="y" THEN RUN "menu03.bas"
380 END
```

```
1000 '
1010 *R.INDEX
1020 '
1030 OPEN "2:INDEX1.dat" FOR INPUT AS #2
1040 FOR I=1 TO 2 STEP 0
1050  IF EOF(2) THEN I=3:GOTO 1090 ELSE DM=DM+1
1060  IF FLAG=1 THEN GOTO 1080
1070  INPUT #2,CO:GOTO 1090
1080  INPUT #2,A(DM)
1090 NEXT I
1100 CLOSE #2
1110 RETURN
2000 '
2010 *SET.INX
2020 '
2030 FOR I=1 TO N
2040  FOR J=1 TO INDEX
2050   IF B(I)=A(J) THEN B(I)=J:J=INDEX+1
2060  NEXT J
2070 NEXT I
2080 RETURN
3000 '
3010 *DAT.PRT
3020 '
3030 CLS
3040 LOCATE 10,5:PRINT "ｲﾝｻﾂ ｼﾏｽ"
3050 LOCATE 10,6:PRINT "ﾌﾟﾘﾝﾀ ｦ ｾｯﾄ ｼﾃｸﾀﾞｻｲ"
3060 LOCATE 10,8:PRINT "Hit any key if OK"
3070 Y$=INKEY$:IF Y$="" THEN GOTO 3070
3080 FOR I=1 TO N
3090  GET #1,B(I)
3100   LPRINT TAB(5);PN$
3110   LPRINT TAB(5);AD$
3120   LPRINT TAB(15);BA$
3130   LPRINT
3140   LPRINT TAB(10);NM$;TAB(30);" ｻﾏ"
3150  FOR J=1 TO 5:LPRINT :NEXT J
3160 NEXT I
3170 CLOSE #1
3180 RETURN
```

次に示すのは、この章で扱ったデータをプリンタに出力したものです。あなたもためしてみてください。

また、いろいろ集計データや資料などを出力するルーチンを作り、宛て名書きをしたあとすぐに出力させるようにして、窓つき封筒を利用して発送すれば、かなり実用化できると思います。

```
030
bbbbbbbbbbbbb
            456-78

     ﾄﾖﾄﾐ ﾋﾃﾞﾖｼ                    ｻﾏ




000
xxxxxxxxxxx
            0-0-0

     ｵﾀﾞ ﾉﾌﾞﾅｶﾞ                    ｻﾏ




101
aaaaaaaaaa
            1-2-3

     ﾄｸｶﾞﾜ ｲｴﾔｽ                    ｻﾏ

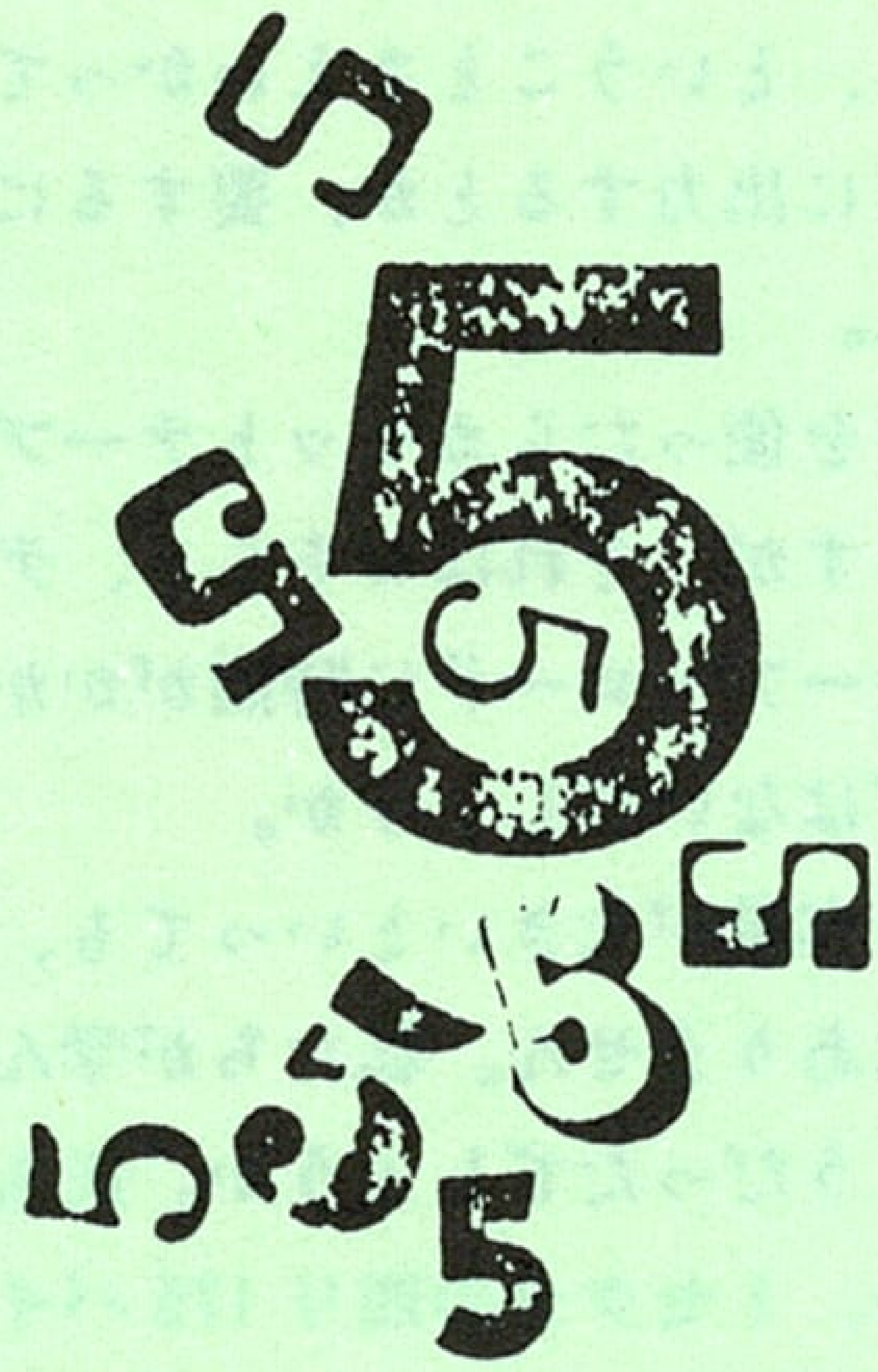



680
dddddddddd
            9-8-76

     ｱｹﾁ ﾐﾂﾋﾃﾞ                     ｻﾏ
```

第5章

ランダムファイル[II]



5・1　ディスクの有効利用

第4章の宛て名書きプログラム、いかがでしたか？　ディスクとのデータのやりとり、ご理解いただけたでしょうか。

ランダムデータファイル管理の基本は、おおよそ第4章に述べたとおりです。数値データを扱いませんでしたが、MKI$、MKS$、MKD$を使ってディスクに保存し、使いたいときにCVI、CVS、CVDで数値に変換する、ということさえわかってしまえば、あとは加減乗除をしてプリンタに出力するとか、要するにN88-BASICでの操作と変わりありません。

ディスクを使ったらカセットテープはもう使う気にならない、とよく耳にしますが、それはおそらく、データを保存する容量がきわめて大きい、セーブ・ロードに時間がかからない、といったあたりに理由があるのではないでしょうか。

しかし、容量が大きいといっても、できるだけ有効に利用するにこしたことはありません。私たちが学んだ、前章でのデータファイルの使い方はどうだったでしょうか。氏名、郵便番号、住所、番地で81セクタを使い、1セクタの残り175バイトは未使用でした。ずいぶんムダな使い方をしていたわけです。

この章では、ディスクの有効利用をテーマに話を進めますが、できたら話につながりがあったほうがよいと思いますので、前章のプログラムを例にとって、一部を変更し、一部ステートメントを追加し、ご説明したいと思います。

ムダのないデータ設計

前章では1セクタ内に1名分のデータを書き込んでいました。

今度は下の図のように3名分のデータを保存することにしましょう。1名で85バイト使いますから、3名で255バイト、つまり1バイトだけ未使用ということになります。ほぼ100パーセント活用です。

氏　名(20バイト)	郵便番号(6バイト)	住　所(39バイト)	番　地(20バイト)
氏　名(20バイト)	郵便番号(6バイト)	住　所(39バイト)	番　地(20バイト)
氏　名(20バイト)	郵便番号(6バイト)	住　所(39バイト)	番　地(20バイト)

←1バイト未使用

さて、１セクタ内のデータフォーマットが決まりましたから、まず問題になるのは**フィールドの割り当て**です。

前章で、

```
FIELD #1,20 AS NM$,6 AS PN$,39 AS AD$,20 AS BA$
```

のように割り当てていましたので、同じ考え方をして次のように定義すればよいのでしょうか？

```
FIELD #1,20 AS N1$,6 AS P1$,39 AS A1$,20 AS B1$
FIELD #1,20 AS N2$,6 AS P2$,39 AS A2$,20 AS B2$
FIELD #1,20 AS N3$,6 AS P3$,39 AS A3$,20 AS B3$
```

一見だまされてしまいそうな感じですが、この場合、３名分の各項目が同じFIELDを共有していることになり、これでPUT、GETをすると、データが混同してしまいます。

ですから、バッファ上に３名分のフィールドを確保したいのでしたら、すべてコンマで区切って次のようにつなげて定義すればよいのです。

Ⓐ

```
FIELD #1,20 AS N1$,6 AS P1$,39 AS A1$,20 AS B1$
         ,20 AS N2$,6 AS P2$,39 AS A2$,20 AS B2$
         ,20 AS N3$,6 AS P3$,39 AS A3$,20 AS B3$
```

そしてこれを少し変形して次のようにしても、それぞれの氏名、郵便番号、住所、番地に応じたフィールド幅を確保することができます。

Ⓑ

```
❶ FIELD #1,  0 AS D$,20 AS N1$,6 AS P1$,39 AS A1$,20 AS B1$
❷ FIELD #1, 85 AS D$,20 AS N2$,6 AS P2$,39 AS A2$,20 AS B2$
❸ FIELD #1,170 AS D$,20 AS N3$,6 AS P3$,39 AS A3$,20 AS B3$
```

これはどういうことかと申しますと、まず❶のFIELD文で、**先頭のＤ＄は０ですから、先頭にＮ１＄が入り順次Ｂ１＄まで設定されます。**次に❷であらためてFIELD文が定義され、まず**85バイトのＤ＄が入り、そのうしろに続いてＮ２＄～Ｂ２＄が設定されます。**

ダミー変数を使う

ここに登場する**Ｄ＄はダミー**で、実際には何も利用されないものです。そしてこの85バイトという数は、❶のFIELD文で定義されている変数のフィールド幅を合計したものです。

つまり、❶でＮ１＄～Ｂ１＄のフィールド域が確保され、❷ではそのフィールド幅をもったダミーを踏み台にしてＮ２＄～Ｂ２＄のフィールド域が決められています。ということはバッファ上ではＮ１＄からＢ２＄までのフィールド域が作られたということになります。

同じように、❸のFIELD文によって、N１＄からB３＄までが確保されたことになります。

❶〜❸の過程で、ダミーのD＄は0、85、170バイトと変化しますが、最終的には170バイトが残ります。

しかしこのD＄、まったくのダミーですから、プログラムでGET、PUTさえしなければよいのです。

さて、これでだいぶFIELD文の下準備ができてきました。

要は、ダミー変数を使えば、そのうしろに希望のフィールド域を確保できるということを理解していただけばよいと思います。

ここでもう一度前に戻って、Ⓐのfield文を次のように配列を使って書き換えてみましょう。

```
FIELD #1,20 AS N$(1),6 AS P$(1),39 AS A$(1),20 AS B$(1)
        ,20 AS N$(2),6 AS P$(2),39 AS A$(2),20 AS B$(2)
        ,20 AS N$(3),6 AS P$(3),39 AS A$(3),20 AS B$(3)
```

そしてⒷも書き換えてみます。

```
FIELD #1,  0 AS D$,20 AS N$(1),6 AS P$(1),39 AS A$(1),20 AS B$(1)
FIELD #1, 85 AS D$,20 AS N$(2),6 AS P$(2),39 AS A$(2),20 AS B$(2)
FIELD #1,170 AS D$,20 AS N$(3),6 AS P$(3),39 AS A$(3),20 AS B$(3)
```

もうおわかりと思います。

いくらなんでも同じ形のFIELD文を3行も長々と書くのはちょっとナンセンスですね。FOR〜NEXT文でもう少しすっきりさせたいものです。

そうです、次のように表せますね。

```
200 FOR I =1 TO 3
210  FIELD #1,85*(I-1) AS D$,20 AS N$(I),6 AS P$(I),
                         39 AS A$(I),20 AS B$(I)
220 NEXT I
```

ずいぶんさっぱりした形に表現できました。

このように、バッファをいくつかのレコードで使用する場合には、文字変数をたくさんつらねたりしないで、FOR〜NEXT文によって処理できるのです。それに、ダミーの考え方、これも大切ですからぜひ体得してください。

さて、前章では1セクタに1レコードを書き込んでいましたが、この例では1セクタに3レコードを書き込めます。すると、PUT、GETはどうしたらよいのでしょうか？

GET、PUTするセクタ番号、そのセクタ内で何番目のレコードかがわかればよいわけですから、インデックスファイルのレコード番号を３で割った商、余りを利用することによって求められます。この辺は具体例で学ぶことにしましょう。

これから、前章で作ったプログラムを一部変更・追加して、１セクタに３名分書き込めるようにしようというわけですが、一番大きな変更点はFIELD文です。

そして、いちばん大切な追加点はセクタ番号、セクタ内でのレコード番号を計算するステートメントです。

すでにでき上がっているプログラムに手を加えてということですから、条件文など、多少まわりくどい点もやむをえないと思います。

しかし、個人で、自分の仕事に合ったプログラムを作ろうとするとき、やはり最初に小さなプログラムを作って、次に、あんなこともしたい、こんなこともしたいと欲がでてきて、変更・追加をくりかえして、プログラムをふくらませていくのです。

それでは、プログラムの手直しにかかりますが、順を追っての説明はやめ、修正後の全プログラムリストをかかげ、追加点は緑色の文字で、修正個所は緑色のアミで示すことにしました。

データ書き込みプログラム

```
10 ' file name ==> dmail1.bas
20 ' 1984.1.25
30 ' random data write
40 ' save "dmail1.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
115 DIM NM$(3),PN$(3),AD$(3),BA$(3)
120 CLS
130 '
140 *OPNF
150 OPEN "2:dmail1.dat" AS #1
155 FOR I=1 TO 3
160 FIELD #1,85*(I-1) AS DUMMY$,20 AS NM$(I),
        6 AS PN$(I),39 AS AD$(I),20 AS BA$(I)
165 NEXT I
170  IF LOF(1)<>0 THEN GOSUB *COUNT:GOTO 200
```

```
180 OPEN "2:index1.dat" FOR OUTPUT AS #2
190 REC=1:GOTO *DAT.INPUT
200 OPEN "2:index1.dat" FOR APPEND AS #2
210 '
220 *DAT.INPUT
230 LOCATE 10,5:PRINT "ﾃﾞｰﾀ ﾉ ﾆｭｳﾘｮｸ ｦ ｼﾏｽ"
240 FOR I=1 TO 5000:NEXT I
250 CLS
260 PRINT "ｺｰﾄﾞ No. ｦ ﾄﾞｳｿﾞ (END=9999)"
270 INPUT "    ==> ";CD
280  IF CD=9999 THEN GOTO 490
290 PRINT "ﾅﾏｴ ｦ ﾄﾞｳｿﾞ"
300 INPUT "    ==> ";A1$
310 PRINT "ﾕｳﾋﾞﾝﾊﾞﾝｺﾞｳ ｦ ﾄﾞｳｿﾞ"
320 INPUT "    ==> ";A2$
330 PRINT "ｼﾞｭｳｼｮ ｦ ﾄﾞｳｿﾞ"
340 INPUT "    ==> ";A3$
350 PRINT "ﾊﾞﾝﾁ ｦ ﾄﾞｳｿﾞ"
360 INPUT "    ==> ";A4$
370 PRINT
380 INPUT "   .  OK ﾃﾞｽｶ  ( y/n )";ANS$
390 IF ANS$="n" OR ANS$="N" THEN GOTO 250
400 '
410 *DAT.WRITE
412 SEC=(REC-1)¥3+1 ←――――――――セクタ番号の計算
414 P=REC MOD 3 ←――――――――セクタ内でのレコード番号を計算
416 IF P=0 THEN P=3
417 IF P=1 THEN PUT #1,SEC
418 GET #1,SEC
420 WRITE #2,CD
430 LSET NM$(P)=A1$ :LSET PN$(P)=A2$
440 LSET AD$(P)=A3$ :LSET BA$(P)=A4$
450 PUT #1,SEC
460 REC=REC+1
470 GOTO 250
480 '
490 *F.CLOSE
500 CLOSE #1,#2
510 '
520 CLS
530 LOCATE 8,5:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
540 LOCATE 8,6:INPUT "  ( y/n )";ANS$
550 IF ANS$="y" THEN RUN "menu04.bas"
```

```
560 END
1000 '
1010 *COUNT
1020 OPEN "2:index1.dat" FOR INPUT AS #2
1030 FOR I=1 TO 2 STEP 0
1040  IF EOF(2) THEN I=3:GOTO 1070
1050   INPUT #2,CO
1060   REC=REC+1
1070 NEXT I
1080 REC=REC+1
1090 CLOSE #2
1100 RETURN
```

前章のプログラム「DMAIL1. bas」をロードして、色アミの部分を修正し、緑色のステートメントを追加入力してください。

●修正・追加が終ったら新ファイル名でセーブするのを忘れないようにしてください。

115行のDIM文は、FIELD文の定義に使う変数のための配列ですが、これはなくてもかまいません。配列を宣言しなければ、自動的に10個分は確保されますが、このように、使用する大きさだけ宣言しておけば、メモリの節約になると考えたからです。

155行から165行のフィールド設定、もうおわかりですね。

412行から416行で、セクタ番号(SEC)、セクタ内でのレコード番号を決定しています。

セクタ番号(SEC)は、レコード番号(REC)から1を引いた数を3で割り、その商に1を加算して計算します。

セクタ内でのレコード番号（1番目のレコードか、2番目のレコードか、3番目のレコードか、ということ）Pは、インデックスファイルでのレコード番号(REC)を3で割り、その余り(0か1か2のいずれか)によって決めます。1セクタに3レコード入るわけですから、余りが0であれば、3つ目のレコードになりますから、416行でP=3としています。余りが2のときは、414行でPに代入された、そのままの値です。余りが1のときは、セクタ内の1番目に書き込まれるわけですから、418行のGET文を実行するためにはPUTしておかなければなりません（Input past endエラーになります)。

417行のPUT #1, SECはGOTO420に書き換えても同じです。

418行のGET文はなぜ入れておくかといいますと、450行で1セクタ分まるまるPUTするわけですから、ここでGETしてすでにファイルに書き込まれている同じセクタ内のデータをバッファ上に取り出しま

せんと、450行のPUT文で、消えてしまいます。

たとえば、あるセクタの第1、第2レコードはすでに書き込まれていたとします。そして今回3番目のレコードを入力しました。このとき、このセクタをGETして第1、第2レコードもバッファ上に取り出しませんと、バッファ上の第1、第2レコードに割り当てられた変数には何も入っていませんから、この状態でPUTすると、第3レコードだけが書き込まれ、第1、第2レコードはクリアされてしまいます。このミスはよくありますので注意してください。

460行ではレコード番号を1進めています。

次にデータ修正のプログラムに移りますが、変更・追加個所は、いま説明したようなところです。プログラムを追ってみてください。

データ修正のプログラム

```
10  '  file name ==> dmail2.bas
20  '  1984.1.25
30  '  random data correct
40  '  save "dmail2.bas"
50  '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
115 DIM NM$(3),PN$(3),AD$(3),BA$(3)
120 CLS
130 '
140 OPEN "2:dmail1.dat" AS #1
145 FOR I=1 TO 3
150 FIELD #1,85*(I-1) AS DUMMY$,20 AS NM$(I),
        6 AS PN$(I),39 AS AD$(I),20 AS BA$(I)
155 NEXT I
160 '
170 GOSUB *R.INDEX:FLAG=1
180 DIM A(DM):DM=0:GOSUB *R.INDEX:INDEX=DM
190 '
200 LOCATE 5,5:FLAG=0
210 PRINT "ドナタ ノ シュウセイ デスカ (END=9999)?"
220 LOCATE 7,6:INPUT "code number";CD
230 IF CD=9999 THEN GOTO 300
240 LOCATE 10,8:INPUT "OK デスカ ( y/n )";ANS$
250 IF ANS$="n" OR ANS$="N" THEN CLS:GOTO 200
270 GOSUB *DAT.CORRECT
280 CLS:GOTO 200
```

```
290 '
300 CLOSE #1
310 '
320 CLS
330 LOCATE 8,5:PRINT "ﾎｶ ﾉ ｼｺﾞﾄ ｦ ｼﾏｽｶ"
340 LOCATE 8,6:INPUT "    ( y/n )";ANS$
350 IF ANS$="y" THEN RUN "menu04.bas"
360 END
1000 '
1010 *R.INDEX
1020 '
1030 OPEN "2:index1.dat" FOR INPUT AS #2
1040 FOR I=1 TO 2 STEP 0
1050  IF EOF(2) THEN I=3:GOTO 1090 ELSE DM=DM+1
1060  IF FLAG=1 THEN GOTO 1080
1070  INPUT #2,CO:GOTO 1090
1080  INPUT #2,A(DM)
1090 NEXT I
1100 CLOSE #2
1110 RETURN
2000 '
2010 *DAT.CORRECT
2020 '
2030 CLS
2040 FOR I=1 TO INDEX
2050  IF A(I)=CD THEN FLAG=1:REC=I:I=INDEX+1
2060 NEXT I
2070 IF FLAG=0 THEN GOTO 2250
2072 SEC=(REC-1)¥3+1
2074 P=REC MOD 3
2076 IF P=0 THEN P=3
2080 GET #1,SEC
2090 LOCATE 2,3:PRINT "ﾀﾀﾞｼｲ ﾃﾞｰﾀ ｦ ﾄﾞｳｿﾞ"
2100 LOCATE 4,4:PRINT "ｼｭｳｾｲ ﾅｹﾚﾊﾞ RET ｷｰ"
2110 PRINT
2120 PRINT NM$(P):INPUT "   ==> ";A1$
2130  IF A1$="" THEN A1$=NM$(P)
2140 PRINT PN$(P):INPUT "   ==> ";A2$
2150  IF A2$="" THEN A2$=PN$(P)
2160 PRINT AD$(P):INPUT "   ==> ";A3$
2170  IF A3$="" THEN A3$=AD$(P)
2180 PRINT BA$(P):INPUT "   ==> ";A4$
2190  IF A4$="" THEN A4$=BA$(P)
```

```
2200 LSET NM$(P)=A1$:LSET PN$(P)=A2$
2210 LSET AD$(P)=A3$:LSET BA$(P)=A4$
2220 PUT #1,SEC
2230 RETURN
2240 '
2250 BEEP:BEEP:BEEP
2260 LOCATE 8,5:PRINT "ｿﾉ ｶﾀ ﾊ ﾄｳﾛｸ ｻﾚﾃｲﾏｾﾝ"
2270 FOR I=1 TO 5000:NEXT I
2280 RETURN
```

印刷プログラム

```
10 ' file name ==> dmail3.bas
20 ' 1984.1.25
30 ' address print out
40 ' save "dmail3.bas"
50 '
100 CONSOLE 0,20,0,0:WIDTH 40,20
110 DEFINT A-Z
115 DIM NM$(3),PN$(3),AD$(3),BA$(3)
120 CLS
130 '
140 OPEN "2:dmail1.dat" AS #1
145 FOR I=1 TO 3
150 FIELD #1,85*(I-1) AS DUMMY$,20 AS NM$(I),
        6 AS PN$(I),39 AS AD$(I),20 AS BA$(I)
155 NEXT I
160 '
170 GOSUB *R.INDEX:FLAG=1
180 DIM A(DM),B(DM):DM=0:GOSUB *R.INDEX:INDEX=DM
190 '
200 LOCATE 10,4:PRINT "ｱﾃﾅ ﾉ ｲﾝｻﾂ ｦ ｼﾏｽ"
210 LOCATE 10,5:PRINT "input code number"
220 LOCATE 10,6:PRINT "    (END=9999)"
230 LOCATE 10,8:INPUT "   ==> ";CD
240 IF CD=9999 THEN DIM S(N):GOTO 300
250 LOCATE 10,8:PRINT "                "
260 N=N+1
270 B(N)=CD
280 GOTO 230
290 '
300 GOSUB *SET.INX
```

```
301 FOR I=1 TO N
302 S(I)=(B(I)-1)¥3+1
303 B(I)=B(I) MOD 3
305 IF B(I)=0 THEN B(I)=3
307 NEXT I
310 '
320 GOSUB *DAT.PRT
330 '
340 CLS
350 LOCATE 8,5:PRINT "ホカ ノ シゴト ヲ シマスカ"
360 LOCATE 8,6:INPUT "    ( y/n )";ANS$
370 IF ANS$="y" THEN RUN "menu04.bas"
380 END
1000 '
1010 *R.INDEX
1020 '
1030 OPEN "2:index1.dat" FOR INPUT AS #2
1040 FOR I=1 TO 2 STEP 0
1050  IF EOF(2) THEN I=3:GOTO 1090 ELSE DM=DM+1
1060  IF FLAG=1 THEN GOTO 1080
1070  INPUT #2,CO:GOTO 1090
1080  INPUT #2,A(DM)
1090 NEXT I
1100 CLOSE #2
1110 RETURN
2000 '
2010 *SET.INX
2020 '
2030 FOR I=1 TO N
2040  FOR J=1 TO INDEX
2050   IF B(I)=A(J) THEN B(I)=J:J=INDEX+1
2060  NEXT J
2070 NEXT I
2080 RETURN
3000 '
3010 *DAT.PRT
3020 '
3030 CLS
3040 LOCATE 10,5:PRINT "インサツ シマス"
3050 LOCATE 10,6:PRINT "プリンタ ヲ セット シテクダサイ"
3060 LOCATE 10,8:PRINT "Hit any key if OK"
3070 Y$=INKEY$:IF Y$="" THEN GOTO 3070
```

```
3080 FOR I=1 TO N
3090  GET #1,S(I)
3095  P=B(I)
3100   LPRINT TAB(5);PN$(P)
3110   LPRINT TAB(5);AD$(P)
3120   LPRINT TAB(15);BA$(P)
3130   LPRINT
3140   LPRINT TAB(10);NM$(P);TAB(30);" ｻﾏ"
3150  FOR J=1 TO 5:LPRINT :NEXT J
3160 NEXT I
3170 CLOSE #1
3180 RETURN
```

メニュープログラム

```
10 ' file name ==> menu04.bas
20 ' 1984.1.25
30 ' Menu program to print address
40 ' save "menu04.bas"
50 '
100 CONSOLE 0,25,0,0:WIDTH 40,25
110 DEFINT A-Z
120 CLS
130 '
140 READ MENU
150 DIM M$(MENU)
160 FOR I=1 TO MENU
170 READ M$(I)
180 NEXT I
190 '
200 LOCATE 7,4:PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,5:PRINT "|ｱﾃﾅ ｲﾝｻﾂ ﾌﾟﾛｸﾞﾗﾑ No.2|"
230 LOCATE 7,6:PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270 LOCATE 8,7+I:PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13:INPUT "input job number ";NO
```

```
310 LOCATE 8,15:PRINT M$(NO)
320 LOCATE 18,16:INPUT "OK (y/n)";Y$
330 IF Y$="n" OR Y$="N" THEN CLS:GOTO 200
340 ON NO GOTO 360,370,380,390
350 '
360 RUN "dmail1.bas"
370 RUN "dmail2.bas"
380 RUN "dmail3.bas"
390 CLS
400 LOCATE 15,7:PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
410 END
1000 '
1010 '/// data area ///
1020 '
1030 DATA 4
1040 DATA ﾃﾞｰﾀ ﾉ ﾆｭｳﾘｮｸ･･･････1
1050 DATA ﾃﾞｰﾀ ﾉ ｼｭｳｾｲ････････2
1060 DATA ｱﾃﾅ ﾉ ｲﾝｻﾂ･･････････3
1070 DATA ｵﾜﾘ･･････････････････4
```

162ページの印刷プログラムに、次の2行を追加しておくとさらに充実します。230行で入力されたコードが登録されているものかどうか確認するステートメントです。現在のままですと、間違ったコードが入力されても、そのまま続行されてしまいます。

```
241 FOUND=SEARCH(A,CD)
242 IF FOUND=-1 THEN BEEP:BEEP:CLS
    :LOCATE 10,5:PRINT "ｶﾞｲﾄｳ ｺｰﾄﾞ ﾅｼ !"
    :FOR I=1 TO 5000:NEXT:CLS:GOTO 210
```

ここではSEARCH関数を使っています。これは整数配列の中に、ある整数があれば、その要素ナンバーを返し、なければ－1を与えます。印刷プログラムでは、110行で整数宣言がなされていますからよいのですが、もしこれがないときは配列名に%をつけなければなりません。

第6章

マスターメニュープログラム



6・1 マスターメニュープログラム

これで本書も終りです。

簡単な英単語練習プログラムにはじまって宛て名印刷プログラムまで、シーケンシャルファイル、ランダムファイルを通して、データの扱い方について勉強してきました。

最初はちょっととっつきずらい面もあったかもしれませんが、ここまで進んでこられたみなさんは、ずいぶんなじんでいただいたのではないかと考えています。

さて、私たちがこれまでに作ってきた4つのプログラム

1　英単語練習プログラム
2　電話帳プログラム
3　宛て名印刷プログラム（No. 1）
4　宛て名印刷プログラム（No. 2）

には、それぞれメニュープログラムがついていて、メニューを選択することにより、データの修正・追加など、希望の処理をすることができました。

ここでは、さらに上の4つのプログラムを管理するマスターメニュープログラムを作っておくことにしましょう。

このマスターメニュープログラムをRUNさせると、いま述べた4つのプログラム名が表示され、希望のプログラムナンバーをキーインすると、さらにそのプログラムのメニューが表示されるというように、仕事が大変やりやすくなります。

つまり私たちがやるべきことは、システムを立ち上げて、このマスターメニューを走らせさえすれば、あとはコンピュータまかせということになります。

システムディスクには**IDセクタを書き換えるためのユーティリティープログラム「SET UP INFO」**があり、これを実行してIDセクタを書き換えれば、システム立ち上げからマスタメニュープログラムの実行まで自動的にやってくれますが**(オートスタート)**、マスターメニュープログラムのファイル名を簡単にしておけば、キーインのわずらわしさもさほどではありませんから、わざわざオートスタートにするまでもないと思います。

もっとも、あなたのビジネスでのシステムが軌道にのり、あとは他人にまかせられるということにでもなれば話は別で、オートスタートにしておくに越したことはないでしょう。

さあ、それではマスターメニュープログラムをキーインしてしまいましょう。

入力が終ったら、これをRUNさせて、いままでのプログラムが正常に走るかどうかテストしてみてください。

```
10 ' file name ==> M
20 ' 1984.1.25
30 ' Master menu program
40 ' save "M"
50 '
100 CONSOLE 0,25,0,0:WIDTH 40,25
110 DEFINT A-Z
120 CLS
130 '
140 READ MENU
150 DIM M$(MENU)
160 FOR I=1 TO MENU
170 READ M$(I)
180 NEXT I
190 '
200 LOCATE 7,4:PRINT "┌";
210 PRINT STRING$(21,"─");"┐"
220 LOCATE 7,5:PRINT "│  ﾏｽﾀｰ ﾒﾆｭｰ ﾌﾟﾛｸﾞﾗﾑ  │"
230 LOCATE 7,6:PRINT "└";
240 PRINT STRING$(21,"─");"┘"
250 '
260 FOR I=1 TO MENU
270 LOCATE 8,7+I:PRINT M$(I)
280 NEXT I
290 '
300 LOCATE 11,13:INPUT "input job number ";NO
310 LOCATE 8,15:PRINT M$(NO)
320 LOCATE 18,16:INPUT "OK (y/n)";Y$
330 IF Y$="n" OR Y$="N" THEN CLS:GOTO 200
340 ON NO GOTO 360,370,380,390,400
350 '
360 RUN "menu01.bas"
370 RUN "menu02.bas"
380 RUN "menu03.bas"
390 RUN "menu04.bas"
400 CLS
410 LOCATE 15,7:PRINT "ｼｺﾞﾄ ｦ ｵﾜﾘﾏｽ"
420 END
```

```
1000 '
1010 '/// data area ///
1020 '
1030 DATA 5
1040 DATA ｴｲﾀﾝｺﾞ ﾉ ﾚﾝｼｭｳ······1
1050 DATA ﾃﾞﾝﾜﾁｮｳ·············2
1060 DATA ｱﾃﾅ ﾉ ｲﾝｻﾂ··········3
1070 DATA ｱﾃﾅ ﾉ ｲﾝｻﾂ No.2·····4
1080 DATA ｵﾜﾘ·················5
```

いかがでしたか、テスト結果は？　うまく動きましたか？

ここで、もう１つだけ、こんなふうにしたらもっとよいのでは、ということを申し上げておきます。

現在マスタメニューで番号を選択すると、それに対応したプログラムメニューが表示され、希望の処理が終ると「ホカ　ノ　シゴト　ヲシマスカ」というメッセージが現れますが、「Y」をキーインしてもマスターメニューへは戻ってきません。

ですから、いままで作ったプログラムのこの部分をマスターメニューをRUNさせるように変更しておくと、もっと使いやすくなると思います。

RUNさせるファイル名を「M」に変更するだけですから簡単ですね。みなさんが自分自分でためしてみてください。

本書の続刊について

この本では一応ランダムファイルまで勉強しましたが、セクタの使い方がまだまだ画一的です。続刊では、もう少し汎用性のあるセクタの使用方法について解説してみたいと思います。

ぜひ、引き続きご利用くださるようお願いいたします。

なお、本書についてのご意見、ご希望などございましたら、ナツメ社編集部パソコン係宛て、どしどしお寄せください。今後の企画に役立てさせていただきます。

ありがとうございました。

PC-8801mkII
データファイル入門 1

編著者　ナツメ社編集部

発行者　田村正隆

発行所　株式会社　ナツメ社
東京都千代田区神田神保町 1 －52(〒101)
電話　03(291)1257(代表)
振替　東京 3 －58661
印　刷　ラン印刷社
製　本　三修紙工

ISBN4-8163-0358-8

PC-8801 プログラミング入門

野々山隆幸著　　1400円

●ズブの素人にも理解できる恰好の入門書です．漢字の使い方も解説してあります．

【内容】 ◆デモプログラムのロード　◆関数計算もできます　◆N88-BASICのスタート方法　◆システムディスクのバックアップ　◆プログラムの作成　◆プログラムのセーブ　◆経営分析プログラム　◆データインプットのプログラム　◆設備投資計画プログラム　◆棒グラフの作成　◆エンド部分とサブルーチンのプログラム　◆利益計画プログラム　◆損益分岐点図表プログラム　◆予算実績差異分析プログラム　◆ＰＣ-8001用プログラムを使うには

PC-8801 ビジネス・プログラミング

布田 治著　　1500円

●ビジネスへの適用の手順を販売管理を例にとってやさしく解説してあります．ディスクベースですので，ディスクをお持ちの読者向けです．ビジネス・ソフトの基本は，この本にすべて含まれています．

【内容】 ◆問題解決の定石　◆ディスクシステム「ひらけゴマ」　◆ディスクに何か書き込もう　◆入力プログラムの作成　◆漢字で見出しを作ろう　◆項目の印刷方法アラカルト　◆どう並べかえるか　◆分類プログラムの実行　◆マッチングプログラムの実行　◆印刷文字のいろいろ　◆ランダムファイルってなあに　◆書いたものは読めなきゃウソ　◆更新の「三種の神器」　◆漢字名称はクセモノ　◆インデックスがなきゃランダムじゃないよ　◆分析表の作成　◆いまやグラフィック時代　◆顔は円グラフ　◆最後に笑うのはCHAINくん

PC-8801 アセンブリ言語プログラミング入門

脇 英世著　　1600円

●内蔵アセンブラを使ったマシン語入門です．具体的な短いプログラムをサンプルに解説していますので，大変わかりやすくなっています．数値のソート，文字のソートまで扱っています．

【内容】 ◆BASICとアセンブリ言語　◆予備知識はいりません　◆モニタプログラムの動かし方　◆１と０をひっくり返すこと　◆大きい数をさがすこと　◆16ビットの足し算　◆引き算はこうします　◆和を計算する　◆ASCII符号とはなんだろう　◆同じ文字列かどうかしらべる　◆精度の高い足し算　◆簡単な並べかえ　◆キーワードとジャンプテーブルを使う　◆ＰＣ-8801のBASICを探検する　◆BASICと機械語のプログラムをつなげる　◆16ビットのデータを並べかえる　◆機械語プログラムの呼び出し方

PC-9801 グラフィックス入門

脇 英世著　　1600円

●インテル８０８６を搭載した16ビット・パソコンでのグラフィック入門です．グラフィックスに必要なコマンドをやさしく解説し，色の塗り方，簡単な図形の描き方を短いプログラムを折りこみながら説明しています．

【内容】◆座標系とは何んだろう　◆WINDOWとVIEWの使い方　◆円はどうやって描くのだろう　◆可変後退翼ジェット機を作る　◆ラインスタイルとは何んだろう　◆ゴールデンゲートブリッジを作る　◆タイルパターンを作るためのツール　◆ペイント文で色を塗る　◆物体を回転させてみよう　◆３次元空間における拡大と縮小　◆ワイヤー・フレーム・モデルを作る　◆宇宙船エンタープライズ号を作る

PC-6001 グラフィックス入門

矢野和代著　　1200円

●ピコピコサウンドのゲームばかりではパピコンもかわいそう．これは，ちょっとノーブルに次元を変えて楽しむための本です．CRTディスプレイによるグラフィックスとプロッタプリンタを使ったグラフィックスをあつかっています．

【内容】◆パピコンでＣＧを　◆めくるめくページ　◆色，いろいろ　◆折れてもグラフ　◆パピコン絵本　◆ダイヤモンドの輝き　◆お知恵をはいしゃく　◆ＰＣ-6022はなんでも屋　◆色を変える　◆あて名印刷プログラム　◆プロッタ，それは魔法の手　◆文字のさかだち　◆くるくるくる，目がまわる　◆何がでてくればいいの？　◆交通整理ピーッ！　◆わかりやすく簡単に

PC-6001 プログラミング入門

安立隆一著　　1200円

●パピコン操縦法から簡単なゲーム・プログラミングまでやさしく解説してあります．

【内容】◆スイッチ・オン　◆たし算をやってみよう　◆ついでに三角関数も　◆さあいよいよプログラム　◆LIST命令で，すっきり　◆プログラム，へんしーん　◆じゃま者は消せ！　◆ゲームのプログラムを作ろう　◆パピコンのサイコロ　◆何回であたったかな　◆ダイスゲームとは　◆サイコロの入れかえ　◆パピコンで音楽を　◆和音もできます　◆テンポも変えられる　◆SOUND命令を使ってみよう　◆UFOを射て　◆UFOメインプログラム　◆ビーム砲操作メインプログラム　◆パピコンとジャンケン　◆ジャンケンの手の形を作る　◆勝ち負けの判断　◆パピコンの可能性

PC-8801mkⅡ
データファイル入門[1]

発行———1984年3月1日
編著社———ナツメ社編集部
発行者———田村正隆
発行所———株式会社ナツメ社
郵便番号101
東京都千代田区神田神保町1-52
電話〈03〉291-1257
振替 東京3-58661
〈落丁・乱丁本はお取り替えします〉

定価———1300円

PC-8801mkII

データファイル入門[1]

［目次より］

ナツメ社

定価=1300円

ISBN4-8163-0358-8 C2054 ¥1300E